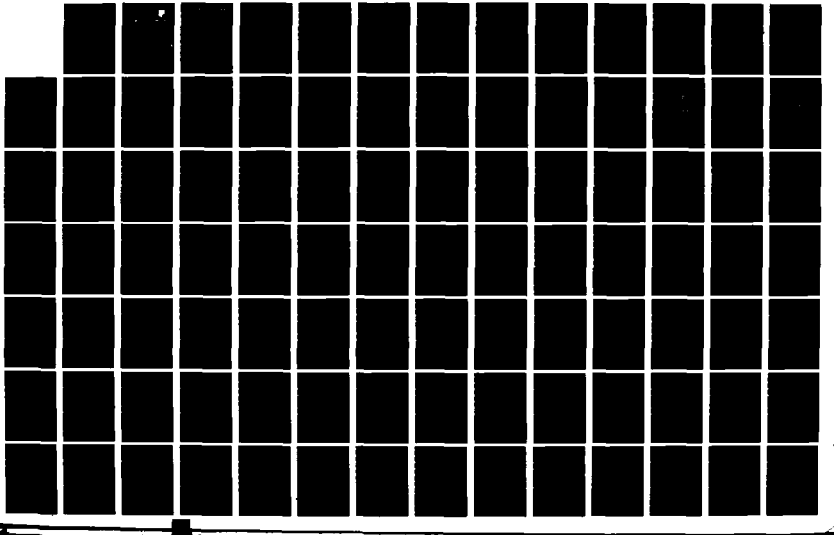
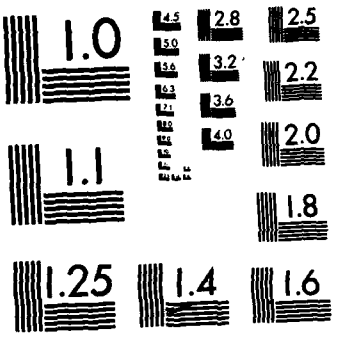
AD-A125 645 AUTOMATIC FEATURE EXTRACTION SYSTEM(U) PAR TECHNOLOGY CORP NEW HARTFORD NY J L CAMBIER DEC 82 PAR-82-19 1/2
RADC-TR-82-200 F30602-78-C-0080

UNCLASSIFIED F/G 20/6 NL





(12)

RADC-TR-82-200
Final Technical Report
December 1982

AUTOMATIC FEATURE EXTRACTION SYSTEM

PAR Technology Corporation

Dr. James L. Cambier

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

RADC-TR-82-200 has been reviewed and is approved for publication.

APPROVED: [signature]

FREDERICK N. KAHRIG
Project Engineer

APPROVED: [signature]

JOHN N. ENTZMINGER, JR.
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: [signature]

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRRE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-82-200	2. GOVT ACCESSION NO. A125645	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AUTOMATIC FEATURE EXTRACTION SYSTEM		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 2 Feb 78 - 31 May 82
		6. PERFORMING ORG. REPORT NUMBER PAR Report 82-19
7. AUTHOR(s) Dr. James L. Cambier		8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0080
9. PERFORMING ORGANIZATION NAME AND ADDRESS PAR Technology Corporation Route 5, Seneca Plaza New Hartford NY 13413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63701B 32050302
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRRE) Griffiss AFB NY 13441		12. REPORT DATE December 1982 13. NUMBER OF PAGES 162
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Frederick W. Rahrig (IRRE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image Processing
Feature Extraction
Pattern Recognition

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The AFES is designed to be a testbed for evaluation of semi-automatic and computer-assisted techniques for automated production flow processes. Its intended input sources included National Sensors and LANDSAT imagery, and its functional capabilities are expandable to permit its use as an experimental testbed for feature extraction. Initial AFES capabilities are applicable to the extraction of planimetric, cultural, and landscape characteristics as required for production of Digital Feature Analysis

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Data (DFAD). The system's hardware configuration provides for input scanning and conversion, image storage and retrieval, interaction with multiple softcopy displays, feature delineation, and output plotting of feature data. The major hardware subsystems include a host processor which performs overall system control, data acquisition and storage, and certain computationally intense procedures: a scanner/plotter subsystem which allows input of image data from film and opaque source materials and generates graphics plots; and a display subsystem which allows direct user interaction with the imagery. The AFES software includes a large collection of system and applications modules which support a wide variety of functions. The operating system supports a multi-user environment, a tree-structure, a complete software control system for system and applications programs, program development aids, documentation aids, and interfaces to all peripheral devices and subsystems. Applications software provided with the AFES supports pixel measurement extraction; pixel classification via statistical pattern recognition; image preprocessing, enhancement and filtering; image warping, resampling, and point positioning; and symbolic image processing via a rule-based inference system.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This document describes the results of the contract entitled Automatic Feature Extraction System, RADC No. F30602-78-C-0080. It is intended to fulfill the requirements for CLIN 0005, Data Item 004.

The AFES contract was performed between 2 February 1978 and 31 May 1982. Under the terms of the effort, a testbed image processing system, including hardware and software, was developed and installed at the Defense Mapping Agency Hydrographic and Topographic Center in Brookemont, Maryland. This report will describe the system and its capabilities.

1.1 SCOPE

The AFES is designed to be a testbed for evaluation of semi-automatic and computer-assisted techniques for automated production flow processes. Its intended input sources include National Sensors and LANDSAT imagery, and its functional capabilities are expandable to permit its use as an experimental testbed for feature extraction. Initial AFES capabilities are applicable to the extraction of planimetric, cultural, and landscape characteristics as required for production of Digital Feature Analysis Data (DFAD).

The system's hardware configuration provides for input scanning and conversion, image storage and retrieval, interaction with multiple softcopy displays, feature delineation, and output plotting of feature data. The major hardware subsystems include a host processor which performs overall system control, data acquisition and storage, and certain computationally intense procedures; a scanner/plotter subsystem which allows input of image data from film and opaque source materials and generates graphics plots; and a display substation which allows direct user interaction with the imagery.

The AFES Software includes a large collection of system and applications modules which support a wide variety of functions. The operating system supports a multi-user environment, a tree-structured file system eminently suited to image processing, modular software structure, a complete software control system for system and applications programs, program development aids, documentation aids, and interfaces to all peripheral devices and subsystems. Applications software provided with the AFES supports pixel measurement extraction; pixel classification via statistical pattern recognition; image preprocessing, enhancement and filtering; image warping, resampling, and point positioning; and symbolic image processing via a rule-based inference system.

The basic AFES capabilities as defined in the original Statement of Work (PR No. I-7-4700 dated 77Feb10) were amended by PR No. I-0-4779 dated 79Oct04. This amendment provided for much of the system's modularity and flexibility, and for most of the specific applications routines which have been included in the AFES.

1.2 BACKGROUND

The development of technologies for exploitation of digital imagery is mandated by DMA transition to all-digital source materials by the late 1980's. A number of research programs preceding and concurrent with the AFES development have addressed the use of digital imagery for the generation of various DMA products.

The RADC Image Processing System (IPS) is a predecessor to the AFES which provides an interactive image processing capability for research related to feature extraction and classification from reconnaissance sensor imagery. The earliest work leading to the development of IPS consisted of integration of a number of pattern recognition software modules in the late 1960's and early 1970's to form the On-Line Pattern Analysis and Recognition System (OLPARS). The system was configured on a CDC-1620 computer with associated peripherals.

In the early to mid 1970's an image-processing front end to provide m ltivariate vector data for input to OLPARS was developed on a PDP-11/20 minicomputer. Software included a custom designed operating system, executive, and large library of application functions most of which were written in assembly language. This system was initially called the Image Feature Extraction System (IFES). A separate effort called Spectral Combination for Reconnaissance Exploitation (SCORE) added multispectral software to IFES, and the combined system was renamed Digital Interactive Complex for Image Feature Extraction and Recognition (DICIFER). This total package was a display-oriented minicomputer system dedicated to developing, testing, and evaluating techniques for imagery exploitation. It was used for processing of black and white and multispectral reconnaissance photography, side-looking synthetic aperture radar imagery, forward-looking infrared imagery, LANDSAT imagery, and several other types of two-dimensional array data. The addition of OLPARS structure analysis and classification logic to DICIFER resulted in a complete pattern analysis capability based upon image data. This system has since become known as the Image Processing System (IPS).

The current IPS hardware configuration is the standard AFES configuration (less the scanner/plotter subsystem), consisting of a PDP-11/70 and PDP-11/34, with mass storage, special processing, and display peripherals appropriate to support the AFES software. Current contractual efforts such as Advanced Pattern Recognition (F30602-80-C-0319) are devoted to enhancement of AFES target recognition capabilities. The AFES control structure is being modified and additional applications software is being added to apply statistical pattern recognition to region measurements. Edge-based image segmentation algorithms are being added, and the AFES Symbolic Image Processor (SIP) is being expanded to accommodate more flexible interaction between symbolic and statistical pattern recognition algorithms.

1.3 TEST RESULTS

This section detail the results of the AFES Final Acceptance testing, which was performed between November 1981 and April 1982. The majority of the testing was performed over five one-week periods, with the test procedures for these periods divided up as follows:

1. Scanner/Plotter Subsystem Testing

2. Feature Measures and Statistical Pattern Recognition. This included commands from the meas and class menus.

3. Symbolic Image Processing, Program Development, and Administrative commands. This group included commands from the sip, prog, and admin menus.

4. Display Commands. This included commands from the disp, itt, init, and misc (on the workstation) menus.

5. Transforms, Input, and remaining Testbed commands. This category included commands from the trans, input, test, and misc (on the host processor) menus.

Commands from the mens menu, the Mensuration Package, were tested separately in conjunction with the testing of the sensor model implementation.

1.3.1 Scanner/Plotter Subsystem

The scanner/plotter subsystem passed all of its performance tests, with the exception of the maximum resolution, which was specified as 80 lp/mm. The highest observed resolution was approximately 72 lp/mm, but it should be noted that this is a highly subjective evaluation. In testing the AFES an attempt was made to estimate resolution conservatively. It is not likely that this resolution limitation will degrade the system's utility.

When the graphics scanner camera was tested, a number of dark vertical lines were observed on the imagery. These were due to specks of dirt on the sensor array which were mechanically dislodged, correcting the problem.

Several minor mechanical and electrical problems were noted and later corrected. The plotter pen cartridge tended to slip in its holder. This was remedied by mounting an external collar on the cartridge where it protrudes from the holder, which prevented it from slipping upward into the holder. The second problem was a tendency on the part of the plotter control logic to reset, disabling the plotter servos, when the pen solenoid was activated. This was caused by cross coupling between the limit switch input wires and the pen solenoid wires. It was corrected by installing an extra shielded twisted-pair cable between the plotter control box and the gantry to carry the pen solenoid signal. Finally, the plotter arrived at DMA with a crack in one corner of its glass top. The glass top was replaced.

1.3.2 Software Testing

The vast majority of the software test procedures were exercised without incident. For each command a test procedure had been written which listed the operator input required to exercise the command, and specified the response expected on the part of the system. For the pixel measurement extractors several small test images were generated, and the output data produced by each measurement was printed. These results were then manually verified using the algorithm on which each program was based. Any exceptions to correct performance were noted on the test procedures, so that they could be corrected and retested prior to the end of the test period.

Just prior to testing of the image transformation commands, such as the Fourier and Hadamard transforms, a hardware failure occurred in the AP-120 array processor. Hence the AP versions of these routines could not be tested. The AP was later repaired and the routines were then tested successfully.

1.4 REPORT ORGANIZATION

The remainder of this report is devoted to detailed descriptions of the major components of the AFES. The sections which follow are available on-line on the AFES as topical documents. A list of available documents may be obtained by entering the command "doc" with no arguments.

Section 2 is an overview of the entire AFES system, and provides a brief summary of its hardware and software structure and capabilities. Section 3 describes the utilization of UNIX system facilities, including software control and on-line documentation. Section 4 is entitled Statistical Pattern Recognition, and describes the various types of pixel measurement extractors and statistical classifiers available on the AFES. A logical continuation of image exploitation is provided in Section 5, which covers the AFES Symbolic Image Processor (SIP), which is a rule based inference system for high level classification of features. Section 6 discusses the AFES Image Processing Language, which is the primary control structure for image exploitation experiments. The warping, resampling, and mensuration software are covered in Section 7, entitled Photogrammetric Software. Finally, Section 8 is a description of the Scanner/Plotter Subsystem. It is derived from off-line documentation of the subsystem, which was built under subcontract by Bendix Research Laboratories. Appendix I lists the user commands available for execution of AFES applications software.

2. AFES OVERVIEW

2.1 INTRODUCTION

This section presents an overview of the Automatic Feature Extraction System (AFES), and serves as a directory to a series of topical documents which describe the theory and implementation of various system components. The purpose and general goals of AFES will be described first, followed by discussions of its overall structure, and finally by a synopsis of the topical documents.

2.2 AFES PURPOSE

The AFES is an integrated hardware/software complex. It is designed as a testbed for applying image processing, photogrammetry, pattern recognition, and artificial-intelligence-derived techniques for semi-automatic map generating and updating. The AFES has been designed as a complete man-machine system for image understanding and efficient receiver of algorithms. The AFES possesses facilities for easily reimplementing, integrating and testing algorithms developed elsewhere, as well as new algorithms. The system is capable of handling both digital and film sources. It also contains elaborate facilities for image input and storage, and can be operated by persons unfamiliar with computers. Accordingly, strong emphasis has been placed throughout its development on such things as modularity, user interfaces, and software support. These goals will be discussed in more detail in the next section.

2.3 AFES GOALS

The AFES design specifications require that it be a multi-user system which is easily modifiable, modular, and independent of image source.

2.3.1 Multi-user Facility

Multi-user implies that expensive resources can be fully utilized by sharing among users. This has been accomplished by use of a work station configuration, in which each user has, for his exclusive use, interactive devices and minimal computational capabilities appropriate to his task. Resources shared with other users, including mass storage devices, special types of processors, and image input devices, are controlled by a central processor which is linked to a number of work stations.

2.3.2 Easily Modifiable

Easily modifiable means that:

- new algorithms can be easily developed and easily incorporated into any part of the system.
- new algorithms can easily use all techniques and algorithms which have been previously implemented on the system.
- new processes can be easily structured from a variety of algorithms and techniques.

2.3.3 Modularity

Modularity implies that:

- Previously programmed techniques are available to new algorithms.
- Processes may be reconfigured from various modules so that these processes can execute on various processors available within the system. Different flavors of processing can be easily developed.

- System design, implementation, maintenance and modification may be clean and efficient.

2.3.4 Independence From Image Source

AFES must be independent from its image source in order to retain compatibility with all present and future image sources which it may be used to exploit. While each new image source may require a different hardware device to digitize the image data and different software modules for image queing and formatting, the result of the input process will be images in standard AFES file format. The AFES file format has been designed for maximum versatility, and accommodates both single channel and multichannel imagery. Eventual compatibility with all available digital imagery is an AFES goal.

2.4 OVERALL AFES STRUCTURE

This section will present more detailed information about the AFES workstation configuration, software system, executive control, program development aids, and applications software.

2.4.1 AFES Hardware Configuration

The AFES hardware can best be described in terms of three main categories; these are the master processor, the workstation configuration, and the scanner-plotter subsystem. The master processor functions as the vehicle for program development, data storage, and many processing operations. The workstation concept provides for a set of dedicated interactive devices for each user. The type of workstation used depends on the operator's task. In general a number of workstations will be linked to one master processor which allocates shared resources among users. The workstation configuration provides the main human-machine interface for the accomplishment of image exploitation. The scanner-plotter subsystem is dedicated to the input/output of source image data, as well as cartographic data generated from processed

imagery. The scanner-plotter subsystem is included in Figure 2-1 as part of the master processor configuration. However because of its unique hardware and importance it will be discussed separately.

2.4.1.1 Master Processor

The master processor is a PDP-11/70 minicomputer with a variety of I/O devices, storage units and processing resources (Figure 3-1). Input images may be provided on magnetic tape, and tape drives are provided for access and copying of image data. The design includes a scanner-plotter subsystem which is linked to the processor via a communication link and a dual-ported disk system so that film, map, or chart data may be digitized and stored on the disk, and utilized by the system as needed. A second large capacity disk system stores source images and intermediate results of image processing functions executed on the master processor. Processing resources include, in addition to the capabilities of the PDP-11/70, a floating point array processor which is used to perform certain types of tasks involving numerical computation on large blocks of data.

Associated with the master processor is the Program Development Station (PDS). The PDS consists simply of a CRT terminal which is linked to the master processor. It is designed for the user who simply wishes to edit and compile programs, and to execute programs on the master processor for which image display output is not needed. The multi-user, time-sharing operating system used for the master processor can accommodate a large number of these terminals without noticeable degradation in response time.

2.4.1.2 Work Station

The hierarchy of work stations provided seeks to match the hardware configuration used with the task to be performed. The types of workstations have been termed the the Full Function Station (FFS) and the previously discussed Program Development Station (PDS).

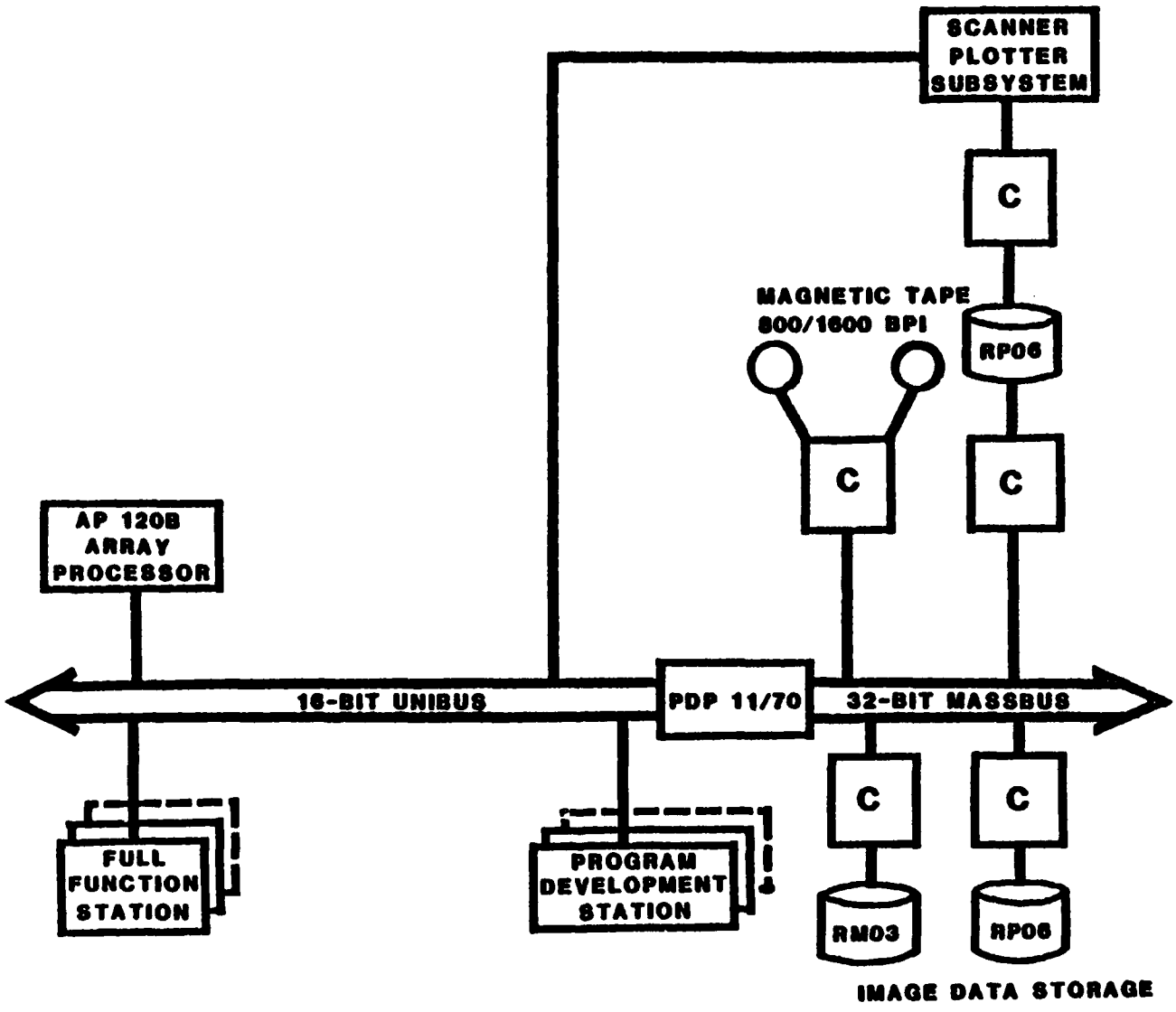


Figure 2-1
MASTER PROCESSOR CONFIGURATION

The FFS (Figure 2-2) provides the full complement of image processing and interaction capabilities. A color display system is included, on which the user may view source imagery or the results of processing operations. Two high resolution monochrome display systems and a stereo viewer are provided to allow display of stereo imagery. Each display system has a trackball, hardware cursors with function buttons, and overlay memory to accommodate operator interaction and display of auxiliary data. A Hewlett Packard Random Scan Display accompanies the FFS. This will be used as a status display to provide the user with relevant information, such as status of background processes and the name of the image that is associated with a particular display channel. In addition, a Dunn color camera system is interfaced to the color display so that hardcopy of source or processed imagery is producible in an efficient, convenient and timely manner. The FFS configuration provides the environment necessary for integrated testing of image processing functions and design and implementation of the types of software systems envisioned for production of digital maps. The display system is controlled by a PDP-11/34 display processor which also provides a minimal processing capability. In particular, operations which require frequent and/or random access to image data, but do not perform complex computations are well suited to execution on the display processor. These may include such things as histogram computation, contrast modification, edge detection, simple geometric transformations, and other preprocessing or enhancement operations. Image data may be transferred to and from the master processor via a high speed parallel data link.

Facilities for operator interaction for the FFS are designed to minimize the knowledge required to use the system. Commands issued by the workstation user may refer to processes which are executed on the master processor or the display processor. To simplify operation incoming commands are automatically sorted by the display processor's command interpretor. Those which run on the display processor are executed immediately, while others are transferred to the master processor's command interpretor.

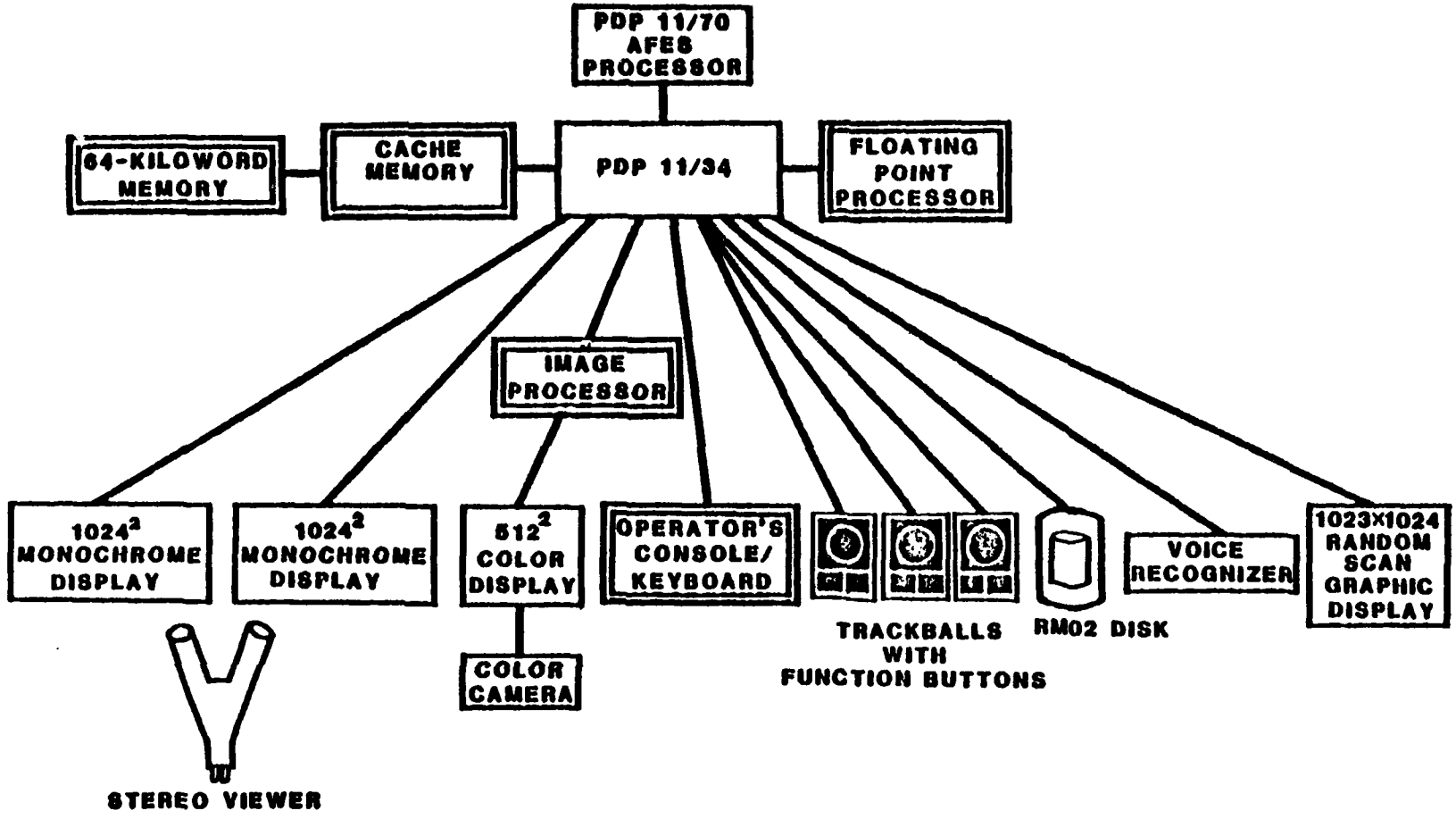


Figure 2-2
FULL-FUNCTION STATION CONFIGURATION

In general, programs which require operator interaction are executed on the display processor. In some cases a single command may start a process on the display processor which will interact with the user to obtain input data or parameters, then start a "batch" type process on the master processor to perform a computation using the user's input data. Most user interaction occurs via trackballs, cursors, and pushbuttons.

An additional interactive device is a voice recognizer, which may be trained by each operator to interpret simple vocal commands and issue the appropriate character strings to the display processor. This allows the operator to enter commands when both hands are occupied controlling trackballs.

2.4.1.3 Scanner-Plotter Subsystem

The scanner-plotter subsystem consists of the scanner viewer, plotter scanner, A/D converter, and PDP 11/34 controller with dual floppy and RP06 disk drives. The scanner viewer consists of two 9 X 18 inch stages, which have optics for scanning or viewing images directly. The scanner for each stage consists of a 1024 element CCD array with zoom optics and rotation capability. It will allow a scan of a 1024 by n pixel image with spot size ranging from 5 to 30 microns continuously. The stages can move at different rates in both x and y directions , allowing skew, rotation, positioning and scale change of the scanned image when coupled with the other features. The grey scale repeatability is one part in 256.

The plotter-scanner is a high accuracy plotter with CCD camera mounted in the pen gantry for scanning of map data. The system will scan opaque or transparency maps. The scale and rotation of the map scanner is fixed with spot size being 80 microns. The plotter can utilize pen scribe. The scan time for the scanner viewing is 8 seconds for a 1024 by 1024 pixel image, while the scan time for the plotter-scanner is about 17 seconds.

The requests for scans are normally initiated from the PDP 11/70 master processor.

2.4.2 UNIX Operating System

The UNIX operating system has been used to provide multi-user time sharing capability. UNIX provides a convenient file structure which supports independence from the image source, an important AFES feature. The UNIX operating system was also designed to support modularity and ease of development. The "Programmer's Workbench" (PWB) version of UNIX has been used for AFES. PWB/UNIX provides the following features particularly important in its application to AFES:

- A hierarchical file system.
- A flexible, easy-to-use command language.
- Ability to execute sequential and background processes.
- The Ned Editor-- a powerful text editor.
- Flexible document preparation and text processing systems.
- Extensive software control capabilities.
- A high-level programming language conducive to structured programming (C).
- The other programming languages LISP and FORTRAN.
- Powerful system I/O routines.

A number of these features will be described in more detail below.

2.4.2.1 UNIX File System

The PWB/UNIX file system consists of a highly uniform set of directories and files arranged in a hierarchical tree structure. Each node in the tree is either a file or a directory; if it is a directory it may have branches to lower level nodes. If one considers a node in the directory tree to be a directory called "dname", then entries in this directory are referred to by a "pathname", for which the entries in dname would be "dname/name1", "dname/name2", etc. Here "name1" and "name2" may be directories or files. The UNIX file system has as its root node a directory containing names of a large number of other directories, each of which contain a hierarchy of other directories and files. This provides a systematic organizational structure. Basic features of the file system are:

- Simple, consistent naming conventions. Names may be absolute or relative to any directory in the tree.

- Mountable and de-mountable file systems and volumes.

- File linking across directories.

- Automatic file space allocation and de-allocation transparent to the user.

- Flexible file and directory protection modes. Directories and files are uniquely associated with a particular user. Both image and user files are coded as to access privileges, with the code indicating read, write, and execute privileges to the file owner, a specified group of users, or to all users. This access control provides file protection and assumes an important role in the UNIX software control system.

- Facilities for creating, accessing, moving, and processing files, directories, or sets of these in a simple, uniform way.

- Treatment of each physical I/O device, ranging from interactive terminals to main memory, as a file, allowing uniform file and device I/O.

2.4.2.2 UNIX Shell

The UNIX command language, called the Shell, is used to implement the AFES "Image Processing Language", which controls file access and user processes, and greatly simplifies execution of image processing functions. The capability for background processes provides for a smoother process flow in execution of statistical pattern recognition routines on images, since some processes, such as classification, can be run in the background while the operator is using the terminal for other routines.

The UNIX Shell language also allows the user to define a wide variety of variables which may be used to simplify command structure. For instance, a user wishing to use the paradigm support software for statistical pattern recognition may define a "working image" with which he wishes to experiment. The pathname of this image is saved as a variable which may be accessed by Shell routines which use the working image as input. Thus the user need not specify a possibly long and complicated pathname each time he executes a command which operates on the working image. Two files, .afesinit and .envinit, set up the proper AFES environment for a particular user, by initializing many of the necessary shell variables.

2.4.2.3 System I/O Routines

UNIX provides standard input and output files which are used whenever possible. The user may specify any file or terminal to be used as standard input or output, or may transfer the output of one program directly to the input of another using these standard I/O facilities.

2.4.2.4 AFES Software

The AFES software has been designed to support the features outlined in Section 3. This involves a discussion of the AFES file system, programming access and aids, and applications software.

2.4.2.5 AFES Commands

It is appropriate to mention AFES commands at this point as they will be referred to in the sections which follow. A more detailed discussion can be found under Applications Programs.

AFES commands are executable files which can be written in the Shell command language or "C". They are organized in "menus" according to the function they perform, and constitute components of the AFES "Image Processing Language". It should be noted that most menu commands with the exception of those in the "meas" section of the menu are shell files. The topical document IPL describes the Image Processing Language in detail.

Command language routines, written in the Shell command language, may in turn start up other system and application programs. These routines may be written in LISP, FORTRAN, "C", or shell. This capability greatly simplifies command string structure, since the command language routine can execute the proper sequence of executable modules based upon a simple set of flags and arguments provided by the user.

2.4.2.6 AFES File System

The AFES file system is based on the standard UNIX file system, and many AFES features are achieved through careful organization and implementation of file structures. As in UNIX, the AFES file system also makes use of a root directory "/", appropriately called the root. Entries in the AFES root directory include, among others, working directories for temporary storage of

image files during processing sessions such as "/w", "/u", which branches down to personal user files, "/usr", which contains system routines, and "/tmp", which contains temporary files. The root node directories most important to AFES are the /u, and working directories. The /u directory will be described in sections which follow.

2.4.2.6.1 AFES Directory

The AFES utility and applications programs are all contained in a user directory which has the pathname "/u/afes". There are a large number of directories under /u/afes; they include the following:

cmd	Program development commands
bin	AFES administrator commands
incl$z	Include files for the current testbed system, release no. $z
bin$z	Master processor command language (Shell) routines, release no. $z
system	System utilities such as control commands for the interprocessor link
lib$z	Master processor library routines, release no. $z
obj$z	Master processor executable modules, release no. $z
smlib$z	Workstation (PDP-11/34) library routines, release no. $z

smobj$z	Workstation (PDP-11/34) executable modules, release no. $z
smbkg$z	Files which are to be run in background at the workstation, release no. $z
sccs	All files placed under AFES control via the "addfile" command
smbin$z	Workstation command language (Shell) routines, release no. $z

Library routines are subroutines used by many programs, which are combined with the calling program to make an executable module. Object modules are compiled versions of main programs. Separate library and object directories are provided for the master processor and the workstation processor. While the workstation routines are maintained by the same software control system as the master processor routines, they are compiled in a slightly different way due to differences in the capabilities of the processors.

The topical document "afeslayout" provides a more detailed description of the layout of directories for the AFES testbed. Successive updates of the AFES software, incorporating new programs and changes to existing programs, give rise to new release numbers. "$z" is a Shell variable which contains the current release number. Hence "/u/afes/lib$z" is the pathname to master processor library routines for the latest AFES release.

2.4.2.6.2 User File Directory

Each user has a personal directory containing programs under development. These are usually maintained under the "/u" root node. While this directory may contain only names of various files, it more often contains names of other directories which divide the user files into categories.

2.4.2.6.3 Image File Directory

The image directories lead to a tree of image files. The sequence of directory nodes in the working image directories are designed to be a highly organized record of all processes which have been executed on an image. This directory structure may be reviewed with the "examine" command, which allows the operator to interactively examine the directory structure. Automatic restart capability is provided by storage of all status information in the file structure so that a particular processing environment can be automatically invoked when a user logs onto the system.

2.4.2.7 AFES Programming Aids and Practices

The goals of modularity and easy modifiability are attained through careful structuring of programs, attention to consistent documentation, utilization of system, library and interface routines, standard image formats, and strict software control.

2.4.2.7.1 Program Structure

AFES applications programs are written in a way which allows maximum flexibility and versatility. Standard software interface routines are used whenever possible, such as the "automatic window" code, described in a later section. Subroutines are written to provide as much application independence as possible, so that they can be used by a large number of programs, thus minimizing the number of subroutines.

2.4.2.7.2 Software Interfaces

One common type of user program anticipated in use of AFES as a test-bed system is the measurement extraction routine which uses as its input the intensity of a single pixel, or perhaps the intensities of pixels contained within a small window surrounding a single pixel. The user is usually

concerned largely with the routine which operates on the pixels within the window, and would rather not have to worry about the mechanics of moving the window throughout the image. Routines of this type may perform, for example, smoothing, edge enhancement, or texture description operations. To meet this need AFES provides a number of include files, referred to collectively as the "automatic window code", which may be inserted in the user window processing code. This automatic window code interprets the program parameter string to obtain names of input and output files, sets up memory allocation for input and output image data, sets up the line-by-line and point-by-point loops which move the window through the image, and performs all necessary data conversions to input and output data. Thus anyone wishing to add new measurement routines has little more to do than to code the algorithm for a single window in the image and insert the appropriate include files. This allows for expandability in the system. Considering the number of computations, window code also executes rather quickly. Typical times for a 512 by 512 image range from under one minute to twenty minutes.

2.4.2.7.3 Utility Subroutines

The AFES subroutine libraries include a large number of functions which, although they are more likely to be used by more advanced programmers, save considerable programming time and help eliminate duplication of effort, and support the modular design of AFES. These routines have applicability in the areas of file handling, display interaction, error handling, and numerical operations, and include the following:

file	locate and open image header and data files
data	convert output data from a given type to an arbitrary format as specified in output image header

input	perform I/O for single lines of image data
matrix	perform matrix addition, multiplication, and inversion
display	initialization of DeAnza display registers; cursor and trackball interaction
error	standard routines for printing error messages.

2.4.2.7.4 Standard Image Format

The increasing variety in types of image sensors available for acquisition of mapping information has lead to a concomitant increase in image formats. AFES achieves a good deal of its versatility be reducing image data from all types of sources to a single, standard image format. Thus the image processing operations are independent of image source, and can accommodate both current and future forms of imagery.

2.4.2.7.5 Software Control

AFES has an extensive source code control system which is maintained by the AFES Executive. This system utilizes two UNIX components, the Source Code Control System (SCCS) and the "Make" command, to assure system integrity.

2.4.2.7.5.1 SCCS

SCCS maintains a record of all changes which have been made to a program's source code, making it possible to reconstruct any earlier version of a program at any time. Each time a user modifies one of his programs, he is required to provide a short description of the reason for the change. This generates a historical record of a program's evolution. The AFES user directory (/u/afes) contains source code for each program which is part of the AFES system, thus preventing proliferation of multiple copies of a program

which may or may not be identical.

2.4.2.7.5.2 Make Command

The UNIX "Make" command utilizes file interdependency data in performing recompilation of programs which have been modified. When a program is placed under AFES Executive control, the author specifies the names of all subroutines or other files which the program utilizes. This information is recorded in a "Makefile" placed in the same directory as the program. When a file is modified the AFES Executive can poll all of the Makefiles to find and recompile all programs which depend on the file which has been modified. Usually a recompilation is performed, giving rise to a new "release" of AFES, after a number of programs have been changed.

2.4.2.7.6 Documentation Access and Aids

Consistent and standardization of documentation has been stressed in AFES. This facilitates both development and use of documentation within the system. First a brief description of the types of documentation available will be provided, followed by a look at some of the AFES/UNIX capabilities which are used by programmers in production of documentation. The topical documents "afes_doc" and "prog_dev" describes these features in detail respectively.

AFES provides several standard commands which allow the user to access online documentation for programs under AFES control. As mentioned previously, AFES commands are arranged in menus according to the function they perform. Entering "menu" at a terminal will produce a list of the various menus available. These will be explained further under Applications Software. To gain a list of actual commands in a particular menu, the user should enter "menu <section>".

Every AFES menu command has a short usage information file accessed by executing the command "help <command name>". This command prints out (on the user's CRT) the proper argument sequence for the command, indicating which arguments are optional and which are required. The topical document, "cmd_syntax" describes syntax for AFES commands, and should be referenced prior to using the help command.

An on-line AFES manual is also maintained for menu commands, and may be accessed by typing "man <command name> afes". This provides the user with extensive documentation about the program, including argument list, functional description, files used, and related commands or routines.

Another important source of information is that available through the "doc" command. Document files are present for many AFES files including main programs, subroutines and include files. Menu commands may or may not have an associated "doc". Document files are essential because they are comprised of the type of detailed system information about a program or file necessary for making modifications. A document may be obtained by typing "doc <file name>".

The bottom level documentation is provided with the program source listing. In addition to frequent comments interspersed within the code, a standard documentation section, called a boilerplate, is provided at the beginning of the program. This lists the author's name, the files, subroutines, and macros used, a program description, the compile string, etc. and is most helpful in making subsequent modifications.

Program development aids are provided which allow the user to produce the necessary documentation for his program with a minimum of time and effort spent. The documentation commands "doc" and "man" operate on text files by invoking the UNIX text processing function "nroff". This function performs extensive text formatting operations including automatic numbering of subsections, printing of headings, indentation, etc. The AFES command, "newfile", gives the user a standard format for production of the nroff source

files, so that the documentation can be written by simply "filling in the blanks" in a prestructured document outline.

The source code documentation is written using the same type of prestructured "boilerplate" used for man and doc files. The programmer simply fills in all of the information required, including his name, files used, program description, etc. Only the in-line comments provided with the source code are left up to the programmer's personal style.

A number of other program development aids have also been provided. Aids such as the "add_to_afes" command allow the user to easily add new programs to the AFES software control system, and prompt the user to provide the information necessary to create entries in the proper Makefile. Commands are provided which allow a user to access an existing AFES program, edit it, and record the changes in the appropriate SCCS file, namely the "editfile" and "deltafile" commands.

The result of the documentation support is that all AFES documentation is produced in a consistent format, and the ease of documentation encourages the programmer to produce the documentation concurrent with his development of the program. This helps to avoid the last-minute large-scale documentation efforts which so often plague delivery of large software systems.

2.4.3 Applications Software

AFES is a powerful image processing system. Commands in the menu driven system are an AFES user's key to image exploitation in the testbed environment. These constitute the Image Processing Language. Most commands are accessible at both the FFS and the PDS. However some menu sections are appropriately available at one workstation only. For example, since the color and monochrome display monitors are interfaced to the FFS, display commands are only available at this station. A summary of AFES commands can best be given by briefly examining the menus.

tst - AFES test-bed commands, including many used in statistical pattern recognition.

input - Commands used to enter images into the AFES environment

prog - Program development commands

misc - A list of miscellaneous commands

meas - Measurement extractors

mens - Mensuration commands

class - Classifiers

symb - Symbolic processing commands (accessible on the 11/70 only)

admin - AFES administrator commands (accessible on the 11/70 only)

disp - Display commands (accessible on the 11/34 only)

init - Display initialization commands (accessible on the 11/34 only)

itt - Commands which make use of Intensity Transformation Tables (ITT's) on the DeAnza display (accessible on the 11/34 only)

A listing of the contents of those menus is provided by Appendix I.

2.5 SYNOPSIS OF TOPICAL DOCUMENTS

This section provides a list and brief description of topical documents provided with AFES which are designed to provide a top-level view of system design and capabilities. Each document gives a theoretical treatment of the particular techniques involved, discusses their discrete-data version where appropriate, and describes their implementation in AFES.

Topical Documents

prog_dev	- Program development under AFES
afeslayout	- Layout of AFES directories
auto_wndw	- A guide to construction of code using the AFES automatic window.
cmd_syntax	- This is a description of the syntax for AFES commands
incl_file	- The nature and use of include files
e	- Describes usage of Rand (Ned) editor
keys_e	- Description of control functions for the Rand (Ned) editor
afes_ovrv	- Automatic Feature Extraction System -- An Overview
afes_doc	- An explanation of the documentation available on the AFES

appl_prog - Applications Programming Under AFES

afes_shell - Description of AFES modifications to the shell

dstfile - A description and use of the display status file.

sip - An overview of Symbolic Image Processing on the AFES

afes_ipl - An overview and description of the Image Processing Language usage and syntax.

afes_sys - AFES System Structure describes in more detail the file system software, control system, command structure, and documentation.

im_comp - Image Compression describes various techniques for image compression, and AFES tools for their implementation.

***** The following topical documents are not on-line, and are available in hardcopy only.

Image Enhancement and Preprocessing - A theoretical discussion of image enhancement and preprocessing, followed by its implementation in AFES.

Measurement Extraction and Classification - This describes the AFES support and applications software for statistical pattern recognition.

Warping and Resampling - The Warping and Resampling paper gives a theoretical background for various types of image warping processes, and describes the programs provided in AFES for warping and image registration.

3. AFES SYSTEM STRUCTURE

3.1 INTRODUCTION

This section explains the philosophy behind and implementation of the software system, as structured for AFES, at the system programmer level. The following topics are covered in detail in the succeeding subsections:

- File Structure
- Program development aids
- Software control
- Command structure
- Documentation system
- AFES administrator

A strong motivating force in the development of the AFES system has been to provide a framework from which a user/programmer can test concepts and/or develop programs in a simple manner, while at the same time, to encourage him to follow good software development techniques. These techniques, which include such things as modularity, structured programming, user interfaces, on-line documentation, etc., while highly desirable, can make life tedious for a programmer. So, it is one of the aims of AFES to help him follow these good programming techniques with a minimum of a priori knowledge and effort. This goal has been accomplished, as will be described fully, by the union of many of the commands available within UNIX/PWB into a highly mobile user environment made possible by the UNIX command language, called the Shell.

3.2 AFES FILE STRUCTURE

The motivation behind the design of the AFES file system has been to provide a file handling environment (be it source files, image files, or whatever) in which the programmer is relieved of as many of the cumbersome tasks such as opening and closing files, maintaining directories, insuring file integrity, etc., as is practicable. The AFES file system accomplishes this task by combining many of the UNIX functions such as the shell, SCCS, Make, to form an integrated environment for file handling.

3.2.1 UNIX Files

From the point of view of the user, there are three kinds of files in UNIX: ordinary disk files, directories, and special files.

3.2.1.1 Ordinary Files

A file in UNIX may consist of almost anything a user might want to place in it. Files of text consist simply of a string of characters terminated by an EOF character. If the text were to be displayable, for instance, newline characters would demarcate physical lines in the display.

Binary programs are sequences of words as they will appear in memory when the program is loaded. Image files consist of a string of unsigned 8 bit bytes which describes the intensity of pixels to be displayed on one of the image displays. The structure of a file is controlled by the programs which use them, however, not by the system.

Filenames, as supported under UNIX, must be 14 or fewer characters in length. Under AFES they must be 12 or fewer characters, the first character must be either an upper or lower case alpha character and no control or special shell characters may occur in the name.

3.2.1.2 UNIX Directory Structure

In order for the system to provide linkage between a physical file and its name a directory structure is maintained. The UNIX system has a hierarchical directory structure with the character "/" being the designated separator between levels. The "/" , as a limiting case, refers to the root directory from which all searches for a path name beginning with the "/" start. For example, to find the file named "file.c", whose complete pathname is "/u/mike/bin/file.c", the system would search "root" for a directory named "u". It would then search "/u" for a directory named "bin", etc., until it either successfully locates "file.c" or fails at some point in the search. If the pathname does not begin with "/", then the system will begin the search in the user's current directory which can be determined by the "pwd" command. When a user logs in, his current directory will be set to a unique one assigned to him by the system manager. When he executes the "cd" command his current directory may be changed to the argument given, ie. "cd /u/mike/tmp" or to his login directory if no argument is given, ie. "cd". A new user will be given a login directory ie. "/u/mike" which contains a "bin" directory as a lower node ie. "/u/mike/bin". The importance these directories play in command execution will be discussed in section "5".

3.2.1.3 File Ownership

In UNIX the mapping provided by a directory entry between a file name and the physical file is referred to as a "link". In the case of a normal user, any file he creates will have a link to it which is associated with his login name. He will determine the privileges associated with that file at the time of creation. He may change the read/write/execute privileges via the "chmod" command for three categories of user: owner, group, world. A non-directory file may appear in several directories under possibly different names, each of which constitutes a link of equal status to the file. With the correct privileges the file may be modified by referring to any of the links to it.

Under most circumstances a file would only have one link to it and would be modifiable only by the creator of the file. Only the owner of a file, or the "super user" may change the privileges to a file.

3.2.1.4 Special Files

Under UNIX, program output which might normally be to a disk file may be redirected to any device, such as a terminal, line printer, etc.. UNIX makes this possible by the use of special files. There is a special file associated with each device in the /dev directory. One would write to a device by writing to the special file the same way as to any other file. A similar capability exists for redirection of input.

3.2.2 User Directories

The important directories to the general UNIX user are as follows:

/u - contains directories for each login name
/bin - contains system commands
/usr - contains system routines, commands, and libraries
/tmp - provides directory for creation of temporary files

Two additional directories of importance to image processing in general and AFES specifically are:

/i - permanent storage for image files
/w - temporary storage for image files

3.2.3 AFES Directories

All files which are a part of the AFES software environment, whether source code, "include file", object code etc., are maintained by the AFES administrator in one of the subdirectories of "/u/afes". Two releases of afes directories are maintained by the administrator to support the concept of on-going development in the higher release while maintaining a workable lower release for operator use. When an AFES user logs on, a shell variable "$z" will be set to the number of the release to which he is linked. For each directory below which has "$z" as the suffix to the name there will, therefore, be two directory names which differ only in the substitution value of "$z". In the case of the "afes" 11/70 library directories, for instance, their names might be "/u/afes/lib8" and "/u/afes/lib9", where "9" is the release in which on-going development takes place. The layout of the AFES directories is very important to the total picture of program development, so the following description of the directories will be useful in the sections to follow:

(Each level of indentation represents the next level in the hierarchical directory)

```
u
        afes
                bin
                cmd
                incl$z
                lib$z
                obj$z
                bin$z
                        modules
                        manuals
                                manA_Z
                                mana_l
                                manm_z
                        documents
                                docA_Z
                                doca_e
                                docf_j
                                dock_o
                                docp_t
                                docu_z
                smlib$z
                smobj$z
                smbin$z
                        modules

                smbkg$z
                smcmd


                sccs
                        cmd
                        files
                        manA_Z
                        mana_l
                        manm_z
                        docA_Z
                        doca_e
                        docf_j
                        dock_o
                        docp_t
                        docu_z
                        fort
                        macro
                        make
                        incl
```

The following is a description of the directories and their contents.

- /u/afes/cmd

  This directory contains the program development commands for AFES. A list of these commands is available via the menu command with "prog" as section name. All of these programs are modifiable only by the AFES administrator, and they are all shell command files.

- /u/afes/bin

  This directory contains the AFES administrator commands. Available via "menu admin" and modifiable and executable only by the AFES administrator or other shell routines.

- /u/afes/incl$z

  This directory contains the Makefile and include files for the testbed system. All routines referencing AFES include files do so via -I/u/afes/incl$z in the compile string.

- /u/afes/lib$z

  This directory contains the Makefile and object files necessary to build the AFES 11/70 library. It is built by the administrator via the makelib command and installed in /usr/lib/libafes$z.a . This makes it accessible by using the -lafes$z switch in cc or ld commands.

- /u/afes/obj$z

  This directory contains the object modules necessary to build the testbed executable modules in /u/afes/bin$z/modules. The modules will be remade only by the administrator via afesupdate. The user may modify the source

files via editfile and deltafile , but the object files must be remade by the administrator.

- /u/afes/bin$z

  The directory for all testbed routines. All routines in this directory are intended to be shell commands which may or may not interface to an executable module in the /u/afes/bin$z/modules directory.

- /u/afes/bin$z/manuals

  Since manuals are very release oriented, there is a copy for each release. This directory contains sub-directories each of which contains nroffed versions of the man files. Each directory represents an alphabetic range based on the first character of the manual name.

- /u/afes/bin$z/documents

  Doc files are also release oriented so a copy of each nroffed file This directory contains sub-directories each of which contains nroffed versions of the document files. Each directory represents an alphabetic range based on the first character of the document name.

- /u/afes/bin$z/modules

  These are the actual testbed or measurement extraction command modules which are built by the user. They are only updated by the administrator.

- /u/afes/smlib$z

  This directory contains the Makefile and object files necessary to build the "small" library for the 11/34. It is built by the administrator via the makelib command and installed in /usr/lib/libsmall$z.a . This makes

it accessible by using the -lsmall$z switch in cc or ld commands.

- /u/afes/smobj$z

  This directory contains the object modules necessary to build the 11/34 executable modules in /u/afes/smbin$z. The modules will be remade only by the administrator via afesupdate. The user may modify the source files via editfile and deltafile , but the object files must be remade by the administrator.

- /u/afes/smbin$z

  The directory for all 11/34 shell commands. This directory contains all commands in the testbed menu on the 11/34. These commands are down-loaded to the 11/34 by Make to the directory /u/afes/bin$z. This directory is in the search path for commands for all users on the 11/34.

- /u/afes/smbin$z/modules

  The directory for all 11/34 executable modules. The modules will be down-loaded to the 11/34 by Make to the directory /u/afes/bin$z/modules. This directory is in the search path for commands for all users on the 11/34.

/u/afes/sccs

This directory contains only directories as shown above. All files which are placed under AFES control via the addfile command are stored by SCCS as g-files in one of the sub-directories. The files with no suffix go in cmd; those with ".d" in one of the doc directories; ".m" in one of the man directories; those with ".f" in fort; those with ".s" in macro; those with ".k" in make; those with ".h" in incl; and those with all other suffixes go in files. The Makefiles in the system refer to the g-files

found in the appropriate sccs directory for source code dependency.

- /u/afes/smbkg$z

  This directory contains dummy command names which are down-loaded to the 11/34 by Make to the directory /u/afes/smbkg$z. This directory is in the search path for for all users on the 11/34. The shell knows to execute these commands in background on the 11/70.

- /u/afes/smcmd

  This directory contains program development command which are down-loaded to the 11/34 by Make to the directory /u/afes/cmd. This directory is in the search path for for all users on the 11/34. This allows a user to execute commands in the "prog" section of the menu on the 11/34.

3.2.3.1 Source Code

As indicated in the description of /u/afes/sccs, all source code files (alpha-numeric files) which are placed under AFES control via the addfile command are stored as SCCS read only files in one of the sub-directories of /u/afes/sccs. Once a user places one his programs under AFES control he relinquishes ownership of the file to the AFES administrator. He may, however, modify the file as required by following the procedures delineated in section 4 on Software Control to follow. For on-line information concerning source files the programmer may execute "doc prog_dev".

3.2.3.2 Object code

The term objectcode in the AFES environment refers to the file produced by the compilation of a source code file before entering the link/load stage. This type of file is produced by using the "-c" switch with the C compiler. In the AFES directories libz, objz, smlib$z, and smobj$z this intermediate

stage of compiled code is maintained by Makefiles. The reason for this arrangement will also be explained in section 4.

3.2.3.3 Executable Code

The next level of file we refer to is executable code which is produced when the appropriate object code files and libraries are linked and external references are resolved. These files are maintained in the AFES directories bin$z/modules, and smbin$z/modules. The files in bin$z/modules are not executed directly by the user but may be executed by shell routines such as "classify".

Executable code available only to the AFES administrator differs from the testbed executable code in that the Makefile in /u/afes/bin maintains the commands without the intermediate object code stage.

3.2.3.4 Executable Shell Files

These are alpha-numeric files interpreted by the shell command language. They are maintained by makefiles in certain directories as described above. They essentially provide the user interface to the AFES environment.

3.2.3.5 Libraries

Due to hardware differences between the 11/70 and 11/34, the library routines must be compiled differently for the two processors. Therefore, a library is maintained with two releases for each processor. For information as to what routines are available and how to include a library in the load string, the AFES command listlib(lsl) with the parameter afes or small may be executed. The libraries are maintained in the directories as described above by the makelib command which installs a copy of the archive file libafes$z.a and libsmall$z.a in the /usr/lib directory. This allows a user to link to the library by typing -lafes$z or -lsmall$z in his compile/load string.

3.2.3.6 Includes

The C compiler has a preprocessor which will replace a line indicated by

```
#include "filename"
```

with the contents of the file "filename". In AFES, include files must have the suffix ".h". They are all maintained by makefiles in the incl$z directories as described above. Extensive use is made of includes in the Window Code which will be described in section 3.3.5.

3.2.3.7 Documents

The term doc file refers to a file which is accessed by the doc command. This file is release dependent and maintained in a sub-directory of /u/afes/bin$z/documents as described above. A doc file has a ".d" suffix which is produced when a user executes newfile to produce a doc file for either an AFES file or to create a topical document. The version stored under SCCS control is in NROFF format and is NROFFED before being stored in a one of the documents sub-directories.

3.2.3.8 Manuals

There must be a manual for each command which appears in one of the AFES menus. Manuals are produced via the newfile command which will create a file with a standard boilerplate where the name will be the same as the command with the suffix ".m" added. This file is release dependent and maintained in a sub-directory of /u/afes/bin$z/manuals as described above. The version stored under SCCS control is in NROFF format and is NROFFED before being stored in one of the manuals sub-directories.

3.2.4 Image Files

There are two directories in which image files are stored in AFES: /i and /w. The /i directory is where all permanent image files in afes are stored; and /w is where all feature extraction and classification processing is performed. An image stored in the /i directory may or may not have an AFES image header file associated with it depending on how it was entered into the directory. In order for an image to be used in the AFES environment it must be in AFES standard format. The user may accomplish this task via the enter_image(eni) command. When a user is added to AFES a directory with the name of his login is made in the /w. All image processing output will occur under this directory as will be described in section 5. Every image created in this hierarchical tree will have two files which describe it, a "data" file, and a "hdr" file whose makeup is described by the doc file for image_hdr.h. The following items are included in the header:

A. ver_nr (2-word integer) - The version number is used to flag changes in the header so that programs will not do inexplicable things when using a file with an old header. Hopefully a program can be constructed to update the header if this becomes necessary. The current version number is available under the macro name "VER_NR" in image_hdr.h .

B. depth (2-word integer) - This is the number of lines in the image.

C. width (2-word integer) - This is the number of pixels in each line.

D. type (short) - This describes the organization of the raster data. The following types have been defined:

   1. Raw data, interleave by pixel - all data for a particular pixel is in a contiguous chunk of storage.

2. Feature raster data, interleave by pixel - all data for a particular pixel is in a contiguous chunk of storage. (The distinction between the previous two types is archaic and will be removed from future versions.)

3. Feature raster data, band sequential - all of each component (channel) is in a contiguous chunk of storage. This may be used for files intended for the color display.

4. All else. This will include statistics files.

E. n_chan (short) - This is the number of channels of image data. The maximum allowed value of n_chan is available under the name "MAX_CHAN" in image_hdr.h .

F. format[MAX_CHAN] - (MAX_CHAN 1-byte characters) This string gives the format of the vector elements. Each character in the string may be
'c' (1-byte character data),
's' (1-word short integer data),
'f' (2-word floating point data), or 0 (no more channels).

G. usage[MAX_CHAN] (MAX_CHAN 1-byte characters) - This gives the usage of vector components . Each character in the string may be
'f' (feature),
'n' (classifier node),
'p' (position information),
'g' (general non-image information, such as a covariance matrix),
or
'0' (no more channels).

H. l_marg (short) - This is the number of columns of garbage at left of image.

I. r_marg (short) - This is the number of columns of garbage at right of image.

J. t_marg (short) - This is the number of lines of garbage at top of image.

K. b_marg (short) - This is the number of lines of garbage at bottom of image.

L. t_row (short) - This is the position of the top row of this image within its source image.

M. l_col (short) - This is the position of the left column of this image within its source image.

N. tran_type (1-byte character) - This indicates the type of transformation used to get from the photo coordinates. Recognized values are
'a' (affine),
'p' (projective),
'1' (1st order polynomial),
'2' (2nd order polynomial),
'3' (3rd order polynomial),
'4' (4th order polynomial), and
'5' (5th order polynomial). The maximum allowed number of transformation parameters is available under the name "TRAN_SIZE" in image_hdr.h .

O. tran_parm[TRAN_SIZE] (TRANSIZE 2-word floating point numbers) - These are the parameters of the transformation from the photo coordinates.

P. scale[4] (2-word floating point numbers) - Scale factor for polynomial transformation

3.3 PROGRAM DEVELOPMENT AIDS

Program development aids within AFES are geared to make a programmer's task as pleasant as possible while encouraging him to develop disciplined programming techniques. To aid in the development of a program are UNIX/PWB features: a context video editor, the structured C language, standard error and I/O, system libraries, to name a few. AFES has added many features in the area of program development, such as commands to initialize a file with standard documentation formats (boilerplates) to encourage documentation, additional AFES libraries, "Window Code" (which handles all overhead and file manipulation for feature extractors), a simple program testing environment and easy integration of programs into the AFES environment.

3.3.1 Video Editor

UNIX supports a command-driven line editor called ed which is useful in some applications. In addition to this editor, the RAND context editor is available to programmers via the e command. This powerful editor allows the user to view a full screen of text while editing the file. This makes the concept of filling out preformatted documentation (boilerplates) possible. The editor is used by several of the AFES commands to facilitate documentation of files and other required items. Some useful tools of this editor include support for multiple file windows, global changes, interface to external filters, and many manipulative commands. For detailed information as to its usage, refer to the on-line documents for "e" and "keys_e".

3.3.2 Documentation Format

As mentioned above, all documentation required in AFES is preformatted. The execution of certain AFES program development commands automatically enters the screen editor with the correct boilerplate. The newfile command

brings in the boilerplate for producing a source file in a particular language. The boilerplates for source files are associated with the suffix of the file and are maintained in the /u/afes/bin directory in the following files:

```
afesdoc   - C documentation
featdoc   - C routine to be written using window code
shelldoc  - shell file documentation
lispdoc   - lisp documentation
fortdoc   - Fortran documentation
macrodoc  - assembler documentation
```

After creating the source routine, the user will be asked if he wishes to create documentation. If he answers yes, then a boilerplate for producing the appropriate documentation is entered into the editor. The file /u/afes/bin/manual is for manuals and /u/afes/bin/document is for doc files. The newfile can also be used later to create documentation files.

3.3.3 Subroutine Libraries

The UNIX system provides a number of subroutines to aid in program development. In addition to these subroutines are many which have been added by AFES. The "cc" compiler will search the system "s" library automatically for subroutines. To cause the search of additional libraries the programmer must specify the "-lname" argument, where name is the name of a library to be searched for a subroutine. The following is a brief description of the libraries available, their contents, and the compile string required.

3.3.3.1 System Library

This library contains subroutines which allow the user the most basic entry level into the UNIX operating system. The routines in this library allow him to manipulate the file system, fork and execute process, determine system or file status, set and catch interrupts and errors, etc. In addition

it provides a set of math functions and some basic string manipulation subroutines. No switch is necessary to cause this library to be searched.

3.3.3.2 String/error/sys Library (-lpw)

This library contains three sets of subroutines, the string set, error set, and sys set. The string set is a comprehensive set of alpha-numeric manipulation routines. The error set consists of general-purpose error handling, signal-setting, and signal-catching, and clean-up routines. The sys set of subroutines provides interfaces to system calls that process error conditions and call fatal(). In addition, a few functions which are not available elsewhere are provided.

3.3.3.3 Input/Output library (-lS)

This library is a portable I/O package which offers the convenience of automatic buffer allocation and output flushing where appropriate. It is, in most cases, the preferable library for I/O since it is system independent where the system library is not. It is somewhat less efficient, however, due to buffering of I/O.

3.3.3.4 Write Library (-lwrt)

This library consists of an interface to syswrite that handles all error conditions.

3.3.3.5 Afes 11/70 Library (-lafes$z)

This library consists of some general purpose routines, image processing routines, image header routines, and an error subroutine which should be called by any AFES C subroutine which generates error messages.

3.3.3.6 Afes 11/34 Library (-lsmall$z)

This library consists of subroutines which may be called by routines which are intended to be run on the 11/34. It consists of matrix manipulation routines, display routines, cursor routines, histogram routines, and the error routine.

3.3.4 Include Files

The include files available in AFES consist of, among other things, the files necessary to support the Window Code package. This package enables one to construct a feature extraction routine. In addition there are include files to describe hardware in the system, such as, the DeAnza link and displays. For a list of all the include files one may execute the match_files(mtf) command and enter the pattern: *.h

3.3.5 Window Code

The Window Code (as described fully in section 2 of the topical document, "Program Development Under AFES") enables a programmer to interface an image-processing routine with a minimum of effort. Essentially, one writes a section of code which processes a window of image data and surrounds it with the AFES include files which provide all the linkage to the data and header for an image and perform all file manipulations for him.

3.3.6 Program Testing

All programs written using the Window Code may be tested prior to inclusion into AFES by two methods. The first method is to execute the program with an existing image as input and create an output image. The user

may then display the output image or examine the image data directly. The second method is to move the load module to his personal "bin" directory under $C which is searched by the shell. The user could then enter the name of the command, be it a measurement extractor or classifier, into the method file along with any required parameters (omitting the input and output file names). He could then execute the ext_measures(xms) command in the case of a measurement extractor, or train in the case of a classifier. The system will find the user's version of the file in his directory and execute it as if it were already integrated into AFES. The same technique could be used if it had already become part of AFES but needed modification. No other user would be effected until the changes were complete and the user had entered the changes into the system.

3.3.7 Interface to Programs Under AFES Control

The primary commands enabling a programmer to place a file, subroutine, or command under AFES control, are located in the "prog" section of the AFES menu. The following is a list of these programs as they appear in the menu:

The following commands are available for afes program development:

*** General Informational Commands ***

help	- gives syntax required to execute a UNIX/AFES command based on the syntax description in section 5
afestext (txt)	- gives brief description of AFES file taken from the information entered by the user when the file was first added to AFES via the "addfile" command
man	- gives detailed description of either a UNIX or AFES command. Includes explanation of switches and other parameters.
doc	- gives program documentation for afes files and other topical items describing AFES.
menu	- gives listing of available commands in AFES

*** Basic File Inspection Commands ***

catfile (ctf)	- lists contents of AFES file to standard output
copyfile (cpf)	- get read-only copy of afes file in working directory
lookfile (lkf)	- look at afes file via the "e" editor (read-only)
prntfile (prf)	- print listing of afes file on line printer
listlib (lsl)	- list all available routines in one of the AFES libraries ("afes" or "small")

*** Miscellaneous File Inspection Commands ***

match_files (mtf)	- list all afes files matching a certain pattern
what_file (wtf)	- list current information about an AFES command, object, module, include file, etc. as controlled by the AFES Makefiles.
listdelta (lsd)	- list all deltas (changes) to afes file to include version #, date of change, person making change, and reason for change
differ (dfr)	- list the differences between two versions of AFES file

```
*** File Creation Commands ***
newfile (nwf)      - create source file with afes standard docu-
                     mentation based on the filename suffix
add_doc (adc)      - prepend afes standard documentation to
                     existing source file
afesnroff(arf)     - format text file (used in checking manuals,
                     and documents )
spell              - check spelling in text file
addfile (adf)      - place file under afes SCCS control
add_to_afes (ata)  - place file or command (to be run on 11/70)
                     under afes Make control
add_to_small(ats)  - place file or command (to be run on 11/34)
                     under afes Make control

*** File Modification Commands ***
editfile (edf)     - get copy of afes file for editing in working
                     directory and lock out other modifications
                     to the file
deltafile (dtf)    - record changes to afes file and delete user
                     copy and release the file for further
                     modifications
killedit (ked)     - cancel edit session, delete user copy of
                     file from working directory, and release the
                     file for modification
backup (bkp)       - make a previous version of an afes file the
                     latest one (all previous versions are
                     restorable)
modifytxt (mdt)    - modify brief description of afes file which
                     was entered during execution of the addfile
                     command (this information is used by the
                     AFES libraries in preparing a description of
                     available subroutines).
listfiles (lsf)    - list all afes files user is currently modify-
                     ing or all afes files user has ever created
                     or modified
```

All of these commands operate within the environment of "software control" as will be explained in the next section. They provide the true programmer interface into the AFES system.

3.4 SOFTWARE CONTROL

A significant amount of time can be, and usually is, lost in any software development project due to insufficient control of programs during the development, integration, and subsequent modification cycle. The greater the number of individuals involved, of course, the greater the impact of poor program management. In the AFES environment all source files which are to be incorporated into AFES as an include file, library routine, command, etc., are first placed under centralized control of the AFES Administrator. There are a large number of AFES commands to aid in this process which will be elucidated in a later section.

3.4.1 Software Control

Software control is the mechanism by which AFES accomplishes the task of maintaining continuity in a dynamic programming environment(testbed). The mechanics of software control involve centralized file ownership, monitoring of file editing, and retention of intermediate changes (versions) to files.

3.4.1.1 File Ownership

With any project where multiple programmers are providing input into the system, be it system or application programming, one of the major obstacles to smooth integration is the problem of multiple versions of routines floating around. Invariably someone needs a routine for a specific application. Presuming he hears that someone has written such a program, he often cannot locate the current version. Then if he needs to modify it even slightly he usually creates a new file. The tendency here is for a proliferation of special purpose programs. This does not encourage the programmer to work in a modular environment. If many people have incorporated a routine into their routines then the task of update in case of change becomes enormous. One of the major requirements for system development to proceed at any kind of

reasonable pace in the above scenario is an immense overall knowledge of the system by a few individuals. If for some reason these people leave the project, it may require months for the project to recover.

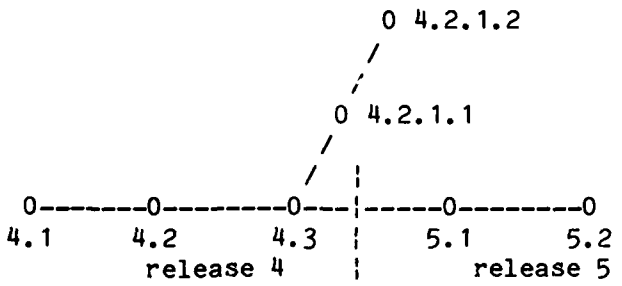
Centralized file ownership is one of the ways AFES avoids the above problems. All files which make up the AFES system are owned by the AFES administrator. They are stored in one of the AFES SCCS directories according to their suffix as described in section 3.2 of this document. All program development commands listed in section 3.3.7 access these files in various ways for the programmer without requiring him to know where the actual source files are located. He can therefore be sure he has the correct copy of the file. He also has many commands at his disposal to determine what routines are available in AFES and gather information about them. He is encouraged to write routines which may be beneficial to others through provision of an easy-to-use interface to the two AFES libraries. Easy access to all AFES files also allows a programmer to explore the software to any depth desired. The user may place a file under AFES SCCS control via "addfile", "add_to_afes", or "add_to_small". The "addfile" command merely places the source file under the afes centralized control in the form of an SCCS file. The other two commands will execute "addfile" if required.

3.4.1.2 Source Code Control System (SCCS)

Having centralized ownership of files, the next step is to control the modification of files. This task is accomplished using the PWB product called the Source Code Control System (SCCS). SCCS stores the original version of a file and all subsequent modifications to it. Any version of the file can be produced by applying the modifications or "deltas" (as they are referred to by SCCS), up to the the version desired, to the original file.

A very important concept in the AFES environment is that of multiple releases. This is accomplished nicely by SCCS. In AFES two releases are maintained at all times. The lower release is intended to be an operational

release at all times with the only changes being the correction of discovered bugs. The higher release is where program development is to take place and may be disabled at times. When a user is added to AFES by the AFES Administrator he is given a read-only file in his login directory with the name ".afesinit". This file is executed upon login and every time a shell is exec'ed. Among other things, this file sets the $z shell variable to a default release # which is carried through by all the program development commands. When a file is entered into the AFES system it is assigned an SCCS Identification string(SID) where the first number is the release to which the user is linked. The second number is the level of the delta, which is always equal to "1" initially. During the course of the project the program development release will become the operational release and a new release, one higher, will be created. One branch from the last delta in a previous release is allowed to develop as a programmer sees the requirement. The following is an example of the evolution of an SCCS file where the file was entered into the system while the programmer was linked to release 4.



The history of the above file shows that after it was initially entered two editfile/deltafile sequences were executed. The program development system then became the operational system thus version 5.1 was made by the AFES Administrator via the nextrls command. The programmer then went through the edit sequence once while in release 5 and two more times while being linked to release 4. The 5.2 version may have occurred before or after the 4.2.1.1 or 4.2.1.2 versions. The listdelta(lsd) command could be used to list all the deltas or only those in the release to which the programmer is linked. The

two releases are totally independent, however.

3.4.1.3 File Editing

When editing a file under AFES control the user always gets the latest version in the release to which he is linked. If he wishes some previous version in the release to be the latest he may execute the backup command. He never loses any of the versions, however. One difference the programmer will notice between the listing of a file he has retrieved via editfile from the same version retrieved via the catfile command will be the absence of any date or time information in the documentation boilerplate. Instead, the programmer will notice some capital letters which are preceeded and followed by the % character. These are recognized by SCCS and the appropriate substitutions for them are made when a listing of the program is requested. They provide the programmer information as to the version #, the date of the listing, date of last update, etc.

3.4.2 System update

After a programmer has made changes to a file which is under AFES control he needs to be sure that the changes are reflected throughout the system. This is accomplished by the AFES Administrator as will be fully explained in section 8. Briefly, a file in AFES which is intended to be used in the system, be it an include file, subroutine, or main routine, must be placed under AFES Make control. The commands to do this are add_to_afes and add_to_small.

3.4.2.1 File dependencies

The concept of file dependency means that one entity in the system is dependent on one or more files in the system. Whenever any of these other files is modified the entity needs to be updated in some manner. The entity

may be no more than a file which contains the contents of two other files; in case of a change to either of the files, the entity is reloaded with current versions of the files. The entity may be,however, a complex executable module which is dependent on a number of include files, library subroutines, and other source files. If any of these files change the module must be recompiled and loaded.

3.4.2.2 Make command

The Make command in UNIX/PWB provides a mechanism by which the system can be kept up to date in a semi-automatic manner. The AFES Administrator is the only one who actually executes this command and he does so indirectly via the afesupdate or tstbed_make(tst). These routines move around in the AFES directories and get an updated copy of the Makefile for the directory and then execute the Make command. There are Makefiles for each of the two releases of AFES maintained. The following is an excerpt from the makefile for the "obj8" directory as an example:

```
#       Makefile
#       will make all the object files in /u/afes/obj$z
#       This is a release dependent makefile and all programs
#       are dependent upon shell variable $z for release #.

CC =  /bin/cc -q -O
FC = /bin/cc -O -I2
LIB = /u/afes/lib8
INCL = /u/afes/incl8
CMD = /u/afes/sccs/cmd
FILES = /u/afes/sccs/files
FORT = /u/afes/sccs/fort
BIN = /u/afes/bin
OBJ = /u/afes/obj8
PROG = /u/afes/cmd
APLIBS = -lV -l
WINDOW = /u/afes/obj8/WINDOW_LIB /u/afes/obj8/WINDOW_INCL

update : makeall

makeall :: WINDOW_INCL
WINDOW_INCL : /u/afes/incl8/image_hdr.h /u/afes/incl8/wndw_init.h
        /u/afes/incl8/wndw_proc.h /u/afes/incl8/wndw_fin.h
        /u/afes/incl8/wndw_buff.h /u/afes/incl8/wndw_read.h
        /u/afes/incl8/wndw_next.h /u/afes/incl8/wndw_write.h
        /u/afes/incl8/wndw_tidy.h /u/afes/incl8/wndw_end.h
        /u/afes/incl8/in_conv.h /u/afes/incl8/out_conv.h
        echo """>WINDOW_INCL

makeall :: WINDOW_LIB
WINDOW_LIB : /u/afes/lib8/if_v_siz.o /u/afes/lib8/if_loc.o
        /u/afes/lib8/if_r_hdr.o /u/afes/lib8/if_w_hdr.o
        /u/afes/lib8/error.o
        echo """>WINDOW_LIB

makeall :: lapl.o
lapl.o : $(FILES)/s.lapl.c $(WINDOW) $(INCL)/image_hdr.h
       $(PROG)/copyfile lapl.c
       $(CC) -c lapl.c -I$(INCL)
       -rm -f lapl.c

makeall :: efilt.o
efilt.o : $(FILES)/s.efilt.c $(LIB)/s_read.o $(LIB)/s_write.o
       $(LIB)/if_open.o $(LIB)/flush_hdr.o $(LIB)/error.o
       $(PROG)/copyfile efilt.c
       $(CC) -c efilt.c -I$(INCL)
       -rm -f efilt.c
```

The purpose of this Makefile is to keep the object code up to date for lapl.o and efilt.o. The Makefile in bin$z will load the object module with appropriate subroutines and libraries. The two are maintained separately for clarity. The lines at the beginning are comments as are any lines preceeded by the # sign. Next are a list of macro definitions which may be substituted in the body of the Makefile with the $(NAME) string, where NAME is the string to the left of the "=" in the macro definition. The first executable line of the Makefile is always made if no argument is given, so it is a convention of AFES to have a dummy line there which is dependent on "makeall" which is dependent on all the items defined in the Makefile. When "make" is executed, all items are checked and updated as required. The single colon indicates dependency and the double colon allows for a continuation of dependencies. In the case of lapl.o, it is dependent on the SCCS source file for lapl.c, the WINDOW macro, and the include file image_hdr.h. If any of these change the make command will execute all of the lines following the line of dependencies up to the next item. In this case it will get the current copy of lapl.c, compile it and then remove the file lapl.c. In the case of efilt.o, it is dependent on the SCCS source file for efilt.c, the afes4 library routines s_read.o, s_write, if_open.o, and flush_hdr.o. The make will proceed as with lapl.o.

3.4.2.3 Placing a Command Under AFES Make Control

As mentioned earlier the two commands which allow one to place an item under AFES make control are "add_to_afes" and "add_to_small". These complex commands are dependent on the highly structured AFES directory layout. These commands allow one to start with either a high level item such as a command or at a lower level such as an include file. If one starts with a command, the SCCS files will be searched for the main routine or shell file as appropriate. If not found the routine "addfile" will be executed automatically for the routine. The user will then be asked questions such as file dependencies and the same process will be repeated for all lower level routines. Finally the Makefiles associated with the command will be modified automatically and the

user will be prompted to add appropriate help and menu entries. If a user wishes to modify the makefile entry for any file he may execute add_to_afes or add_to_small again for that file and the makefile entry will be replaced.

3.5 COMMAND STRUCTURE

There are many command categories in the UNIX/AFES system, with the added complexity of multiple processors. The primary UNIX feature used to solve command problems is the UNIX shell. All command searching is accomplished by the shell, which not only functions as a terminal command interpreter, but which may also take as input a program written in shell syntax. The shell plays an important role in making the AFES command environment clean and easy to use. Currently, the command linkage between the 11/34 display processor and 11/70 host computer is provided by the DeAnza link. The link supports remote display commands and transfer of image data between processors, as well as, execution of commands remotely.

3.5.1 Command Syntax

In the Unix system there are some general principles followed in the syntax used in command generation. However, there are contradictions and some inadequacies which the following AFES syntax will resolve.

All the commands in the afes system are of the Unix form:

command_name parameter...

The following symbols have special meaning in describing the command syntax via help, or man:

< > The angle brackets are for grouping items together and have the highest priority.

[ ] The square brackets indicate to the user than all flags or parameters enclosed are optional as a group.

| | This is the exclusive-or operator which is used to show the operator when only one parameter or group of parameters in a given set is allowed.

() If a comment is required for increased understanding it will be enclosed in parenthesis. They should be avoided if possible.

- The "dash" alone means the command should read from standard input.

-a The "dash" immediately followed by one or more characters represents a flag which the user must type exactly as shown. The terminator for the flag is a space or left angle bracket.

-a<nam> This is the symbol for a flag "-a" followed immediately by the user's input for "nam",ie. -afile1 , where file1 is the name or a file to be processed.

name An item offset with spaces or "< >" is a parameter which the user must enter his response to, such as in the preceeding example.

For the current release all parameters and flags must be entered in the order as shown in the help command. The following examples are given to aid in understanding the syntax rules:

```
cmd parm1 parm2
cmd [parm1 parm2]
cmd parm1 [parm2]
cmd [parm1] parm2     *** error ***
                      (since the command cannot tell
                       whether it is parameter one or two if
                       only one is entered.)
cmd -i -j<name> parm1
cmd [-i<name>] [-j] parm
cmd [<-i name>] parm   (name is required with the -i flag)
cmd -l || -r
cmd <-i [-z<blocks>] [-s<size>] > || <-o [-z<blocks>]>
```

3.5.2 Modifications to the Shell

Several changes have been made to the Interactive UNIX shell to adapt it for use by the afes system. These changes have mostly been made to allow the afes system to maintain shell variables and to use the inter-processor link.

The AFES shells are based on ISC version 3.40 shell but have been upgraded to ISC version 3.45. Additional space has been allocated for parsing (':' operator in expressions) to allow the use of longer strings.

An additional feature has been added to the file scanning procedure in the shell. Besides accepting ':' and ';' as legal delimiters, the afes shell also accepts '%'. When a file to be executed is found in a directory which is delimited by a '%', the program itself is not run. Instead, the name of the program is stored in shell variable I and a program named 'interface' in that directory is executed. The use of this feature in the shell is to specify commands which, when typed on the 11/34, should be handled by executing the program of the same name on the 11/70. The transferring of the command to the 11/70 and taking care of the output is done by 'interface'.

Afes software is dependent on the values of various shell variables including variable $z which contains the current afes release and variable $x which has the path of the current working image. When certain afes programs are run, the values of these shell variables in the top level shell must be changed. This is handled by forcing 'next' commands on the shell. After the shell executes an 'afeslink' command (also known as 'lnk') the shell forces a 'next $C/.afesinit' and after a 'chg_wrk' (also 'cw') the shell forces a 'next $C/.envinit'.

The following commands have been added to the shell to support the afes Image Processing Language(IPL):

cpi photo view frame

cpm method_id

vrb [-]

They allow the changing of current processing image, current processing method and detail of error messages respectively. In detail, the following shell variables are changed:

for cpi,

$s - set to processing photo
$t - set to processing view
$r - set to processing frame
$x - set to $q/$s/$t/$r
$y - set to $x
$p - set to contents of file $x/.spectral

for cpm,

$w - set to processing method

The "vrb" command functions much like "opt" in that it determines the level of detail of afes error messages an operator will receive. It sets the shell variable n to accomplish this. "vrb -" will give the most detailed output and "vrb" the least detailed.

Some other shell variables which are used by afes are variables $D and $H. Variable D is used to store the current date in a form used by some afes programs and variable H is used to store the name of the afes error file used to store error output for programs executed across the link. The afes shell also has an expanded accounting file output. The full command is written to the accounting file rather than just the program name. This will allow users to recover lists of commands they have executed. If a person would like to repeat a series of commands or use a similar series on a different image, he does not have to depend just on memory.

There is a special version of the afes shell called 'transh' which is used for commands sent across the link with the 'tran' command. This shell is always given one command to execute and is not expected to read commands from the terminal. All references to typing out a prompt or typing out the message about mail have been removed along with the check for logout after being idle for 15 minutes (it is never idle). This version of the shell uses a different startup routine (/etc/tran.init) from the usual shell (/etc/sh.init). Also an *extra character is expected prepended to the argument* of the '-c' switch. Rather than let the shell decide if it is the login shell, the shell is considered to be the login shell (variable F set to 1) if the first character in the '-c' argument is a space (or anything except a zero) and not the login shell if the first character is a zero.

3.5.3 Inter-Processor Communications

As mentioned in the preceding section, most of the changes to the shell were in support of the inter-processor link between the 11/34 and the 11/70. The 11/34 is the display station where the user may use the DeAnza displays and any commands available on that processor. All commands in the "tst" section of the menu are executed on the 11/70 based on the user's current working method and image. If he executes one of these commands from the 11/34, the mechanism is such that it is started up in background much as if he

had been on the 11/70 and executed the "run_in_bkg(rib)" command. He will then be notified by mail when the process is complete and may examine the status via the "bkg" command.

The primary routines which accomplish the link are the "tran" and "lnd" commands. The tran command formats the request and assumes the identity of the user executing a "tran -c". On the other side of the link the process is started by the "lnd" command which executes a shell with the same privileges and identity as the initiator of the command. A routine called "get_udata" was added to the system to determine user identity.

The tran command passes a structure block across the link which is decoded by "lnd" for execution. The type of information is: command, parameters, error file, user information. The tran -c command saves the log_info returned via the get_udata call in p->loginfo which is part of the p->name block. If there is a third argument field to the "tran" command , it is taken as a file name on the other side by "lnd" to which stdout and stderr are to be redirected. The lnd routine will fork a version of the shell in /bin/transh which assumes the identity given in the loginfo block via the logpost() function. If there is no fourth argument to the "tran" command then the shell will be started with $F = 1 which is like a login shell. Otherwise, if there is a fourth argument to the "tran" command, the shell will then be started with $F=0 so some of the initialization in the .afesinit will not take place unnecessarily for this command string. To save time, the /bin/transh does a next for /etc/tran.init instead of /etc/sh.init. The two commands which take care of the "tran" details for the user are "modlink" and "shlink". The "shlink" forces the shell to perform like a login shell with all shell knowledge. The "modlink" is faster but has more limited knowledge.

Besides command execution, the tran command also allows the user to transfer a file in either direction across the link; and to see if a file on the other processor is readable, writable, or is a directory.

3.5.4 Command Structure for Master Processor

The commands available to a user depend upon the sh.init shell command which is executed when the user logs in and when a shell is exec'ed. This file has been modified so that .afesinit is executed if found in the user's login directory. For AFES users, it is present and therefore executed. One of the things done by .afesinit is to set up the $X shell variable which is used by the shell to determine which directories to search when a command is entered. The ".afesinit" file is as follows:

```
set z = 8
set X = ":$B:/u/afes/cmd:/bin:/usr/bin:/u/afes/bin$z:"
if $G = 0 set X = "$X/priv:"
set u = "/u/afes/bin$z"
set v = "$u/modules"
set q = "/w/$L"
if $F = 1 then
        set o = c
        if ! -d "$q" mkdir $q
        if ! -d "$q/tmp" mkdir $q/tmp
        set H = $q/tmp/.small
        if "$S" != transh then
                if "$A" = "" then
                        $v/accnt_name != A
                endif
        endif
        next -a $C/.envinit
endif
next
```

As was discussed in a previous section the "$z" shell variable is set to the release to which the user is linked. For afes users the shell searches the following directories for commands: current directory, user's personal bin directory, the AFES program development directory, the system bin directory, and the system users bin directory, and the AFES testbed directory. Given the order of search, it is possible for AFES or any other user to have his own version of a system command such as the "man" command.

The following shell variables are set at login, any time a shell is exec'ed, or after afeslink:

```
$u - afes test bed directory
$v - afes modules directory
$q - user's image work directory
```

The following are only set at login or after executing chg_wrk:

```
$o - default display is color
$H - directory for stdout or stderr for remote commands
$A - account file name for accounting system
```

If the shell were a login shell then ".envinit" is executed after ".afesinit". This also happens any time the chg_wrk command is executed. This is a copy the ".envinit" shell file:

```
if -r $q/.photo then
        set s = <$q/.photo
endif
if -r $q/.view then
        set t = <$q/.view
endif
if -r $q/.frame then
        set r = <$q/.frame
endif
if -r $q/.cur_method then
        set w = <$q/.cur_method
endif
set x = "$q/$s/$t/$r"
set y = "$x"
if -r $q/.spectral then
        set p = <$q/.spectral
endif
next
```

The following shell variables are set by ".envinit":

$s - current working photo
$t - current working view
$r - current working frame
$w - current working method
$x - path to current working image
$y - path to image for statistical processing
$p - spectral type of current working image

3.5.4.1 Shell Interfaces to User Programs

The UNIX shell is really the executive for the AFES system. Being a powerful command interpreter, it allows for the support of various paradigms without the expense of a great deal of the project's development time. Since our control code can, for the most part, be written in this high level language, modifications are more easily made. The two paradigms currently being developed for AFES are: statistical pattern recognition, and Artificial Intelligence. The user has the concept of a current working environment when he logs in. The shell is initialized to reflect this environment as described above. This environment is maintained whether the user is logged in to the 11/70 or the 11/34. Since the input to the shell in this case is the terminal, this is considered the foreground environment. The user may run shell command files which we refer to as the Image Processing Language(IPL) in background which set up a different environment but do not modify the foreground. The IPL will be covered in detail in section 6.

3.5.4.2 Command Types

The program development commands, which were listed in section 3.7, are available to the programmer at all times. The menu command is available to lead the user to commands he may execute.

The following command sections are available via the menu command:

tst	-	Afes test-bed commands
input	-	entering images into the afes environment
prog	-	Program development commands
misc	-	Miscellaneous commands
meas	-	measurement extractors
trans	-	image transformation commands
class	-	classifiers
symb	-	symbolic processing commands
admin	-	Afes administrator commands

The type of commands in each of these menus will described in the following sections.

3.5.4.2.1 tst

These commands allow the user to enter images for experimentation, enter or modify a working method, change working image or method, define measurement sets for training, train a classifier, classify the image, examine the outputs of the process tree, clean up previous output, run a process in background and monitor its status, get runtime information for a command, change processing method or image, and execute symbolic processing. The commands allow for small changes such as the addition of a new region without having to recalculate measurements for existing regions. These shell files execute modules in the /u/afes/bin$z/modules directory as required.

3.5.4.2.2 Input

These commands allow a user to enter an image into the afes format from either the disk or Landsat tapes. This imagery can be either monochromatic or polychromatic; monoscopic or stereo.

3.5.4.2.3 prog

These commands make up the program development commands covered in section 2.

3.5.4.2.4 misc

These commands are a group of general purpose commands which may aid the user in link usage, file examination, or debugging.

3.5.4.2.5 meas

The term "measurement extractor", in the statistical pattern recognition paradigm, is any measurement which is applied to a sample of the data to be classified. For image processing, measurements are not made on individual pixels but on a window of these pixels with the purpose being to reduce noise. The goal of statistical pattern recognition is to accomplish a separation in measurement values between visually distinct areas of the image, which are referred to as classes, ie. trees, field, development, etc. Measurement extraction is based on the user's current working method as will be described in section 6.

3.5.4.2.6 trans

These commands allow a user to execute a number of transforms on the rows and columns of a image, such as: fft, fast-hadamard, etc.

3.5.4.2.7 class

This menu contains classifiers which may be entered into the current working method file and trained if required and executed via the "classify" command.

3.5.4.2.8 symb

These commands are available to the user only after executing "sip", which is the rule-based symbolic processor.

3.5.4.2.9 admin

The AFES administrator has a number of commands at his disposal which enable him to maintain the AFES system.

The following commands are available to the Afes administrator:

```
makelib          - make afes library
nextrls          - creates next release of afes
inmk             - Determines which files are not in Makefile
afesupdate       - updates the afes commands via "make".
tstbed_make(tst) - update the tstbed commands via "make".
adduser          - adds user to afes
                   (login and type /u/afes/bin/adduser)
rmf              - remove sccs file
cleanup          - removes all deltas from an afes file
makemake         - make the makefiles up to date
delta_priv(dtp)  - list users with delta privileges to a file
add_name(adn)    - give user edf/dtf privileges to a file
```

3.5.5 Command Structure for Display Processor

The command structure for the display is similar to that of the 11/70 in that the shell is initialized via .afesinit and .envinit on the 11/34 when the user logs in on that processor. A convention has been followed when a version of a shell command is required on both processors but the code must be different. The convention is to create a new file with the alpha portion of the name preceded by the letters "sm". In the case of .afesinit it would be: .smafesinit. The same convention is followed for some of the directories maintained by the Make command on the 11/70 but whose executable commands are "tran'd" across the link to be executed by a user on the 11/34. This is the

case with the "tst" section of the menu whose commands are stored in "/u/afes/bin$z" on both machines but whose code for the 11/34 is maintained in "/u/afes/smbin$z" on the 11/70. All of this is, of course, transparent to the user with the exception being that the menus on either system will differ slightly.

In addition to the menus available on the 11/70, the user has the following menus available on the 11/34:

disp	-	DeAnza display commands
itt	-	Display itt commands
init	-	Display initialization commands

3.6 DOCUMENTATION

The AFES system maintains multiple levels of documentation on-line, where each level is geared to satisfy specific needs, from command syntax to detailed program documentation. The AFES system provides an interface to the programmer which encourages him to incorporate needed documentation and relieves him from the task of remembering all the levels required. The end result will be a system both usable and modifiable without requiring one to spend an inordinate amount of time just learning the system.

3.6.1 Document Types

There are a number of different types of documentation in the AFES system; and the access to this documentation varies greatly depending on the type. The main documentation for any file will be found internally in the file. The next level of documentation for a file is called a doc file. For a list of commands the user has the menus by section. For each command in the AFES system the user has access to help information. If the help information is insufficient he may refer to the man file for the command. Finally, the user has access to various topical documents to aid in his development of

programs under AFES.

3.6.1.1 Source Code Documentation

The internal source code documentation is required for any source file under AFES control other than ascii tables, such as menus. A different format for documentation exists for each type of file depending on the suffix. A list of the format files for each type was given in section 3.2.

3.6.1.2 Document Files

A doc file is required for every file in the AFES system other than a shell file which appears in a menu. Shell files which appear in a menu will only have a man file associated with them. A main C file which is entered in a menu will have both a doc file and a man file. All other files, such as, tables, subroutines, include files, etc. will have a doc file.

3.6.1.3 Menus

3.6.1.3.1 PDP-11/70 Menus

The following is a list of the menu files and the directories in which they are stored:

/u/afes/incl$z:

menulist - which is a list of the menu sections given to the user if he executes "menu" without a section name.

menupath - a file used to associate a menu section with the physical directory in which the menu resides.

menutst - testbed commands

menumeas - measurement extractors

menuclass - classifiers

menuinput - image input into the afes environment

menumisc - miscellaneous commands

menutrans - image transformation commands

menusymb - symbolic processing commands

/u/afes/bin:

menuadmin - Administrator commands

/u/afes/cmd:

menuprog - program development

3.6.1.3.2 PDP-11/34 Menus

/u/afes/smbkg$z:

menusmlist - list of the menu sections given to the user if he executes "menu" without a section name.

menusmprog - program development commands

menusmtst - testbed commands

menumeas - measurement extractors

menuclass - classifiers

menusminput - image input into the afes environment

menusmmisc - miscellaneous commands

menutrans - image transformation commands

menudisp - display commands

menuitt - itt commands

menuinit - display initialization

The user makes entries to these menus via the add_to_afes and add_to_small commands, but he may also modify any of the menu files via the editfile/deltafile sequence if necessary.

3.6.1.4 Help Files

The following is a list of the help files and the directories in which they are stored AFES system:

/u/afes/bin$z:

helplist - This is the help file for all release dependent commands in UNIX such as found in the menu sections: tst, meas, class.

/u/afes/bin:

cmds - This is the help files for the entire UNIX system in which all AFES program development and administrator commands are kept.

The help command in AFES tries to find the command in the "helplist". If that

fails it executes the system help command which searches the "cmds" file. The "helplist" can be modified by any user via editfile/deltafile if necessary. The "cmds" can only be modified by the administrator.

3.6.1.5 Manual Files

All of the manual files under AFES SCCS control are in NROFF format. They can be modified in the same manner as any other AFES file. An nroffed version of the files, per release, are maintained in the /u/afes/bin$z/manuals directory. The man command will execute the UNIX man command for the default directory if the user does not specify the afes section of the manual as a parameter. If the command is not found in the system manuals then the AFES manuals are searched for the command.

3.6.1.6 Document Files

All of the "doc" files under AFES SCCS control are also in NROFF format. They can also be modified in the same manner as any other AFES file. A nroffed version of the files, per release, are maintained in the /u/afes/bin$z/documents directory. The doc command will cat a copy of the doc file to standard output(normally the terminal).

3.6.2 Documentation Aids

To aid the user in the creation of all of the documentation required by AFES there are a number of techniques employed. These include using a shell interface to bring the correct boilerplate required for all file documentation, be it source code, doc file, or manual; integrating the NROFF formatting routines into the documentation; prompting the user for required menu and help entries when adding a command.

3.6.2.1 Boilerplate

The boilerplates for source files are unique to the language, C, Fortran, or shell but the information required is essentially the same. The boilerplates make use of certain SCCS variables such as date of last modification. The user will fill in the information as required for his routine and delete the portions which are not applicable. The commands which enter him in the Ned editor with a fresh boilerplate are newfile(nwf) and add_doc(adc)

. The add_doc command will prepend the boilerplate to an existing file which was not begun by newfile.

The following is a detailed description of how one would fill out the boilerplate for a C routine:

- FILE NAME

  This is the source file name to be used by the compiler or loader commands. It may contain several entry points but the name of the file may or may not be one of them. For C routines under afes, the suffix must be ".c" with a total length of 12 or less characters.

- VERSION

  The version number is supplied by SCCS via translation of the %I% variable. The following is the meaning of the number:

  release.level.branch.sequence

  For afes the number is limited to one branch off the main trunk.

- DATE OF LISTING

The date of the listing is also supplied by SCCS via translation of the %H%%T% variables.

- PROGRAMMER

The person responsible for this program.

- DATE OF LAST UPDATE

Provided by SCCS via translation of the %G%%U% variables.

- ENTRY POINTS

This is a list of the entry points, ie. main and/or functions contained in this file. There must be at least one.

- INPUT/OUTPUT FILES

This is a list of all data files which are opened by functions within this file. The complete pathname and the r/w mode should be included. A description of the data file layout should be included in a file accessible via the doc command. For example, if a file is opened which follows a particular standard then list the standard by doc name.

- INCLUDE FILES

Include file names are to have the suffix ".h" and be 12 characters or less in length. They are to be loaded by the compiler from the working directory, ie. no pathnames are allowed. The variables which are declared or defined inside an include file are to contain their descriptions within the include file and need not be repeated in the documentation of the C program which references them.

- MACROS

  C provides certain language extensions by means of a simple macro preprocessor. All macros defined by the "#define" compiler control statement will be listed under this section and will contain comments describing the input arguments, if used, and the purpose of the macro.

- GLOBAL DATA STRUCTURES AND VARIABLES

  Global data structures and variables are those which are defined outside any function and are, therefore, available by the same name to many functions. If the variable is defined in another source file, then an extern declaration is required. Otherwise, the variable is defined in this section which is outside the first left brace. There will be only one declaration or definition per line and a description on the same line.

- ENTRY

  This is the beginning of the function dependent documentation which will be required for each main or function entry in the source file. The entry will be listed here as it is in the function definition statement prior to the first left brace of each function. In addition, each argument will be listed beneath the entry with one entry per line and a comment describing it.

- ARGUMENT RETURNED

  For each function there will be an explicit return, or an exit in the case of a main entry. The type of argument returned and its possible values, if significant, will be listed under this heading with one per line. The default type will be integer. The type of argument will correspond to the function type.

- FUNCTION

This should be a one or two line explanation of the function of this entry. The same thing will be used for the addfile command.

- DETAILED PROGRAM DESCRIPTION

The first part of detailed description should be the method of solution or algorithm used by this routine, if applicable. The reason for its existence could also be mentioned here. Any theoretical references would be listed here.

Next, there will be a step by step description as to how the function of this routine was performed. This will be in a structured (if then else) English such as:

```
prepend a "-" to the help argument
while an input line exists
        if the line begins with "-argument"
                write "argument:0
                while an input line exists
                        if the line begins with "-"
                        then take good exit
                        else output the line
                        end of if
                end of while
        end of if
end of while
```

- FUNCTION CALLS

There will be two major headings under functions; File Internal, and File External. Function entries defined in the same source file are placed under the File Internal heading. All functions which require explicit inclusion by file, or by library name ("-l" flag) during compile and load will be included under the File External heading, and will have the library name following the variable.

- COMPILE STRING

  Included at this point will be the actual compile string which would be used to produce an object file of the source file, or an executable module if it has a "main" entry point.

- EXECUTABLE CODE

  This is the main body of the function. A few additional items may appear at this point before the first left brace. The actual declaration of static variables, the function entry itself, and its input arguments(if any), will appear prior to the left brace. After the left brace any local variables will be declared. There will be one per line and each one will have a comment on the same line. They will be grouped by type, ie. integer, char, etc. There is no prescribed order of the groupings.

  Next will begin the actual executable code, which will contain major comments for logical segments and minor comments as required. The end of each function will be an explicit return or exit with value. All actual code generation should use tab indentation corresponding to conditional levels in the code.

3.6.2.2 Formatting Routines

A major contributor to easy documentation is the NROFF formatter. This formatter allows one to easily modify a document, label levels, provide automatic numbering of sequential items, etc. Please refer to the off-line manual for NROFF for details as to how to use it. All of the AFES boilerplates are written so the user only has to fill in the blanks. He only needs to understand NROFF if he wishes to create a more detailed document.

3.6.2.3 Prompts for Menu and Help Entries

Every command which the user wishes to add to the AFES system must be added via the "add_to_afes" or "add_to_small" commands, where add_to_small is for 11/34 commands only. These commands are essentially the same with the difference being in the type of items one might wish to add. The only significance of these commands, in the area of documentation, is to cue the user to add the appropriate entries in the correct menu and help file. This is accomplished by entering the ned editor with the correct file. The user then duplicates an entry replacing the name with the command name being entered.

3.7 AFES ADMINISTRATOR

As mentioned briefly under Software Control, the AFES administrator is the manager of AFES software. He is the one who adds users to AFES, maintains multiple releases of AFES software, and assures system integrity. His task has been made as automatic as possible to reduce human error to a minimum.

3.7.1 Adding AFES Users

When a user is to be added to the AFES environment the AFES administrator must first add the user to the Unix system by creating an entry in the /etc/passwd file and creating a login directory for him. The AFES administrator then logs in as the user and executes the /u/afes/bin/adduser command. This command will bring a copies of the ".afesinit" and ".envinit" files into the user's login directory on both systems and make an entry in the "everyone" file for AFES mail. The next time he logs in he has access to AFES commands and all directories he needs will be created.

3.7.2 Maintaining Multiple Releases

As was mentioned earlier, the AFES administrator maintains two releases of AFES files and commands. The higher-number release is stored in /u/afes/.toprelease for reference by some commands. One additional release (lower) is also maintained. The default release is determined by the $z shell variable which is set when one logs in or is changed when one executes the lnk command. The Administrator receives mail any time a file under AFES make control is modified via edf/dtf. He will update the entire system via the afesupdate command. If no one has modified a file in the lower release(which is usually the case) he will so indicate when executing afesupdate. This has the effect of changing the modification dates of the lower release files effected and obviating any recompilation of files. In the upper release the makefiles are first made via the makemake command. Next the individual AFES directories are remade. The libraries are updated via the makelib command. The 11/34 must be booted and the link established before executing afesupdate or tst since any updated commands are down-loaded to the /u/afes/smbin$z directory on the 11/34.

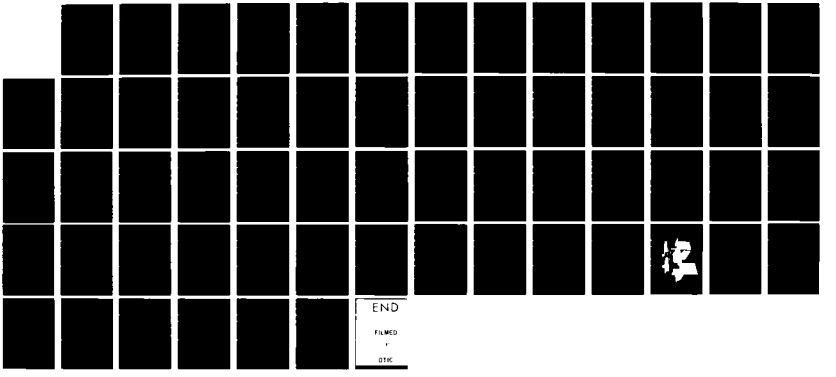
At logical stopping places during the development of AFES the Administrator will decide that the higher release should become fixed as the operational release and the lower release deleted. He may accomplish this task via the nextrls command. He must make sure there are no AFES files being edited before executing this command. This command will do a "get -e" and "delta" for every file in the afes system in the next higher release. It will then make all the directories required and copy all the files into these directories. All users will be notified to link to the new release for development of programs.

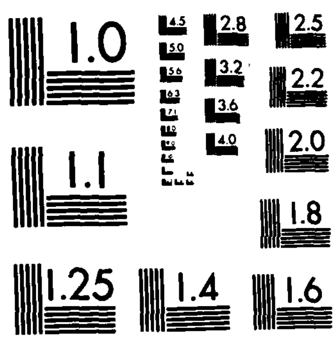
3.7.3 Assuring System Integrity

The AFES framework of SCCS/MAKE has made the job of system integrity as straightforward as possible but the AFES Administrator must still be somewhat involved with the development of programs in the AFES environment. He must make sure that programmers are adding the proper documentation at all levels and that any changes to routines which might affect others are coordinated. If a user forgets to add a routine which makes his loading fail the Administrator must circumvent the mistake so other files which need to be remade can proceed. At the same time he must identify the problem to the programmer. The AFES Administrator is the catalyst which keeps everything running smoothly and efficiently.

AD-A125 645 AUTOMATIC FEATURE EXTRACTION SYSTEM(U) PAR TECHNOLOGY CORP NEW HARTFORD NY J L CAMBIER DEC 82 PAR-82-19 RADC-TR-82-200 F30602-78-C-0080 2/2

UNCLASSIFIED F/G 20/6 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

4. MEASUREMENT EXTRACTION AND CLASSIFICATION

4.1 INTRODUCTION

This section is intended as an introduction to statistical pattern recognition in general, and its implementation in the Automatic Feature Extraction System (AFES) in particular. It proceeds from an introduction of basic concepts to the actual implementation of these concepts within the AFES framework.

Statistical pattern recognition is simply the implementation of the childhood concept of learning by similarity. If a young child is shown several pictures of other animals, he can recognize which of the animals are dogs. He recognizes them in spite of the many differences in details between the example dog pictures and the pictures he is subsequently asked to identify. The child is able to recognize the dogs among the other animals because their pictures are "similar" to the pictures he was initially shown. Hence the child perceives the essential characteristics of what is a dog and what is not a dog. He is then able to differentiate and generalize based on these characteristics.

Statistical pattern recognition is applied to image understanding and feature extraction in a similar manner. There is, however, a significant difference between how the child recognizes images and how the computer learns to recognize images. The child can normally identify for himself the essential characteristics which make a picture of a dog really a picture of a dog. However, in computerized statistical pattern recognition and feature extraction the operator or researcher must identify for the computer the essential characteristics of the area types which are to be identified. Indeed the selection of the characteristics which are to be measured in order to determine the similarity of one thing to another is the critical step.

Once the desired measurements for determining similarity have been selected, classification becomes straightforward.

The subsections which follow describe the processes of measurement extraction and classification in more detail. Selection of measurement and classification algorithms for a particular application normally requires considerable experimentation with test imagery. The AFES provides a convenient mechanism for formulating and executing image exploitation experiments, called the AFES Image Processing Language. It is described in Section 6 of this report.

4.2 MEASUREMENT EXTRACTION

Measurement extraction is the assignment of numerical values to characteristics of objects. To more clearly understand the concept behind measurement extraction, consider for a moment an example not directly related to images. Suppose the problem at hand is to characterize a particular object as to whether or not it has been made of wood. Previously other examples of wood have been shown to the computer program or to an individual who must use measurable quantities for characteristics. Characteristics which come to mind might be shape, density and texture of the object. Shape, while it is a characteristic of the object, would not be good for classification into wood or not wood. Wood can be made into a variety of shapes as can concrete, plaster, plastic and other materials. Density might well be a good characteristic since it is relatively easy to measure, and also would serve to separate wood from most of the other materials listed above. Feeling to the fingertips might also be a good measurement, in the sense of discrimination, except it might be very hard to automatically reduce feeling to a number, even though it is a measure easily distinguished by a human observer. Feeling then is an example of an essential characteristic which is not easily measured by machines for use in statistical algorithms. Measurements for computerized classification must be both meaningful and efficiently implemented.

In the example of wood versus not wood, we are therefore led to density as a good measure to use in classification. In some cases the use of two or more measurements will yield better classification results. Consider the following example. Suppose that we desire to sort finished wood according to species of tree by imaging the wood with a TV camera. At this point we will deal with only two types of wood--birch and ash. How should we proceed? We know for a fact that ash is usually darker in color than birch. Therefore, we could measure the brightness of some samples. If the average brightness value exceeds some value or threshold we will classify the wood as birch; if below the threshold we classify as ash. Once again we are classifying based on a single measurement, yielding a measurement vector, (x1), in one dimensional measurement space. If we want to differentiate even more accurately between birch and ash, we can introduce a second measurement. We also know that ash has a more prominent grain pattern than does birch. Wood grain can be measured through the light to dark transitions in the sample. If this frequency of change per unit distance is above some determined threshold, the object will more likely be ash than birch. Thus, we can use grain as a measurement, x2. Our classification of ash versus birch is now based on a vector X, where X is comprised of two components, x1 and x2. We say that X is a measurement vector in two dimensional measurement space.

We can now be more explicit about how a computer actually performs classification. This explanation will also make clear why selection of appropriate measurements for the classification process is so important. In the example given for classifying a piece of wood as either ash or birch, based on a single measurement, we said that if brightness were above a certain threshold level k, we would classify the piece as birch; otherwise the piece would be classified as ash. This process can be thought of as partitioning the number line into two sets, one set consisting of all numbers above the threshold and the other set containing the remaining numbers.

This is illustrated in Figure 4-1.

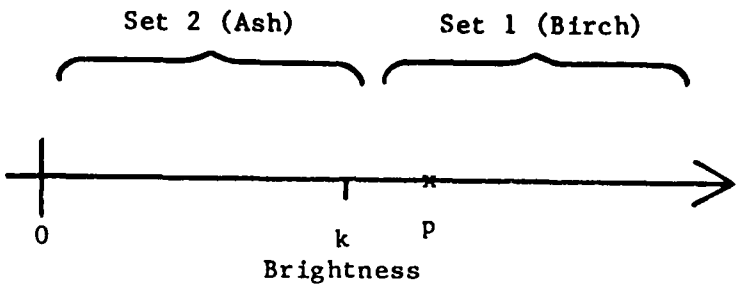


Figure 4-1 One Dimensional Classification

In the example using two measurements, brightness and grain frequency, the classification process may be thought of as partitioning the plane into two sets. Two possible ways of positioning the plane are shown in Figure 4-2.

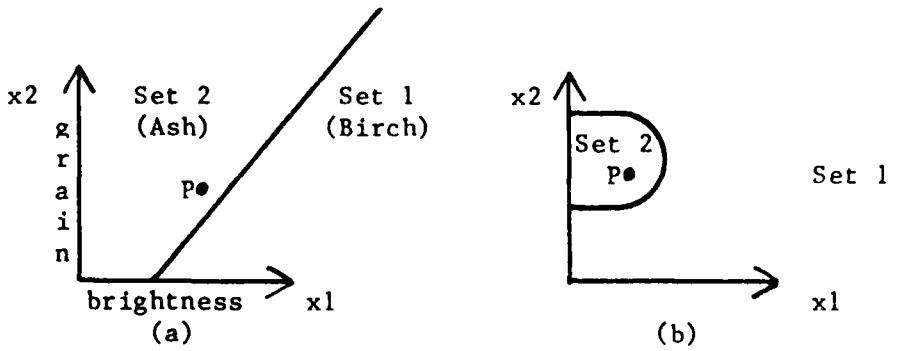


Figure 4-2 Two Dimensional Classification

The examples illustrate the concept of "mapping into measurement space". The measurement space of Figure 4-1 is one dimensional and the measurement space of Figure 4-2 is two dimensional. "Mapping into" measurement space means locating the point in the measurement space which corresponds to the item to be classified. The point in measurement space is determined by the

measurements made on the object. Thus, in Figure 4-1, the point marked "P" corresponds to a piece of wood whose brightness is a little greater than k. The points marked "P" in Figure 4-2(a) and 4-2(b) correspond to a piece of wood whose brightness and grain have approximately equal numeric value.

Choice of measurements defines the measurement space. Choice and training of classifier partition the defined measurement space into sets. The choice of a classifier determines the general shape of the set boundaries (i.e. straight lines, ellipses etc.) and the training precisely locates these boundaries (i.e. slope and intercept of the straight line, centroid and eccentricity of the ellipsoid etc.). Though the examples were of one and two dimensional feature spaces only, statistical pattern recognition often is done in higher dimension spaces. (Thus, lines become hyperplane in the general problem). Classification, per se, of an object is accomplished by determining which set of points in measurement space contains the point corresponding to the object to be classified.

Thus, the central role of measurement selection can be readily appreciated. If the measurements are such that points in measurement space corresponding to the same class of objects cluster closely together and are far (in measurement space) from points corresponding to objects of different class, the precise shape and location of boundaries (i.e. choice and training of classifier) become unimportant. If the measurement space points fail to meet the criteria of same class closeness and different class separation, no classifier can work well.

AFES has a large number of measurement extractors available including edge detectors, texture measures, and transforms. Below is a list of the measurement extractors available in the AFES, and a brief description of their purpose.

acf	- run on the output from mnvar, acf computes the autocorrelation among points
aredge	- area edge detector
athres	- finds the total area of a window above a threshold value
avg	- average (mean) over a window
bestfit	- Gradient of "best plane" fit to 2 x 2 window
bthres	- finds the total area of a window below a threshold value
cnt_peaks	- Hsu texture measure which counts peaks and troughs
contr	- local measure of contrast
cp_hist	- counts peaks of a smoothed histogram of a square window
epq	- equal probability quantized image
ep_smooth	- perform edge preservation smoothing
filt_itt	- apply an itt transformation to a file
gray_crect	- make a correction image from a constant calibration image
gray_match	- apply a correction image to an image
hyperb	- calculates the hyperbolization of a histogram
int_stdv	- finds the standard deviation of the intensities
lapl	- Four point Laplacian
maxcontr	- finds the maximum contrast value of a window
maxint	- finds maximum intensity of a window
median	- median over a window
mincontr	- finds the minimum contrast value of a window
minint	- finds minimum intensity of a window
mitch	- First phase of Mitchell texture measurement
mnvar	- calculates the variance, statistical difference, skewness or kurtosis of an image
moments	- finds the kth moment of a window
pntedge	- point edge detector
rang	- finds the range of intensity values of a window
roberts_x	- Roberts' cross edge detector
sobel	- Sobel edge detector
sumdxdy	- Simple gradient edge detector
sum_peaks	- Hsu texture measure which sums heights of peaks
unshrp_msk	- apply an unsharp mask to sharpen edges of a window

4.3 TRAINING AND CLASSIFICATION

Ultimately characteristics such as those described in the preceding section are used by a classifier to assign the input data to one of a finite number of categories. This process is called classification. In order to determine the essential characteristics of some object, one must first provide samples of that object. The process of giving an example of an object is called training. Training is performed in conjunction with what is termed supervised classification. Unsupervised classification does not require

training, and will be discussed later.

Samples acquired for training must meet two criteria. They must be separable and they should be of informational value. Separability implies that if statistical or quantitative measurements are extracted from each training sample, and the vectors containing these N measurements are plotted in N dimensional space, areas (volumes) representative of each training sample in the plot should be distinct and not overlapping. The N-dimensional space corresponding to the N-measurements is, in most Pattern Recognition literature, termed the "feature space". However, here we will call this space the measurement space to avoid confusion with the use of "features" to describe cartographic features.

Additionally, in practice the training samples must be representative of the population one is looking for. If one wishes to identify coniferous trees in an image, and uses a sample containing many species of trees, the sample is not a true representation and will result in misplaced and misshaped partitions. If training samples are not distinct from each other, or do not present the operator with information helpful in solving his task, subsequent classification results may be poor and impossible to interpret. Thus, in practical cases the selection of training samples is quite important to the entire process.

Once the desired measurements have been determined classification can begin. As previously mentioned, classification is a decision-making algorithm which uses the measurement space derived through training to place the input data into the appropriate category or class. It should be mentioned that several training regions may be defined for a single class. For example in defining a class called vegetation, the user may outline samples of both woodland and farmland as training regions. In cases where two or more sample regions are available, the measurements are combined to form a single vector for the entire class.

How a classifier actually determines the category into which a pixel should be placed varies with the particular classifier algorithm. Those available under AFES will now be discussed.

4.3.1 Mean Nearest Neighbor (mean_nn)

Mean Nearest Neighbor is one of the more simple classification strategies. The mean or average value is determined for each class of training regions based on the output of the measurement extractor(s). An unknown pixel may be classified by computing the distance between the pixel feature value and the mean feature vectors for the various classes. The pixel will be assigned to the "closest" category, i.e. the class having the smallest difference between pixel value and the vector.

4.3.2 Condensed Nearest Neighbor (cnds_nn)

Condensed Nearest Neighbor uses consistent subsets of the training samples to perform a Nearest Neighbor classification. A consistent subset is defined as a stored subset of the training sample which when used as a reference set for the Nearest Neighbor rule correctly classifies all remaining pixels in the sample. An algorithm is used to determine a consistent subset for each training region. In cases where more than one region is defined for a particular class, the subsets are combined. The Nearest Neighbor rule is then applied using the differences among the input pixel versus the subsets to determine the appropriate class.

4.3.3 Mahalanobian (mahal)

The Mahalanobian classifier, as with mean nearest neighbor, determines a mean vector for each class of training regions. In addition the covariance matrix is calculated for the sample regions, providing information as to the dispersion of the data in any direction within a class. The distances between the input pixel and the various classes are computed. These distances are

then normalized according to the spread of the data in the respective class as determined by the covariance matrix. This yields the Mahalanobis distance. The pixel is then assigned to the class to which it is closest using the minimum Mahalanobis distance among the pixel and the various classes.

4.3.4 Multivariate Categorical Analysis (mca)

The Multivariate Categorical Analysis classifier is similar to the Mahalanobian classifier. Compromises have been included to enhance performance, resulting in a cheaper algorithm with respect to execution time. For a detailed discussion of "mca", the reader should reference "Multivariate Categorical Analysis - Bendix Style" by Robert Dye of Bendix Corporation.

4.3.5 Unsupervised Classification (cluster)

An unsupervised classifier is one that does not utilize any training samples. Clustering techniques group the input data into clusters, so that elements in a cluster have a high degree of similarity and elements belonging to other clusters have a large degree of dissimilarity. The technique employed in the "cluster" uses a new measure of similarity called the Mutual Neighborhood Value (MNV). This measure considers the conventional Nearest Neighborhood ranks of two samples with respect to each other. The conventional Nearest Neighborhood rank is computed using the Euclidean distance. The MNV of any two pixels then is the sum of the ranks with respect to one another. If the MNV is equal or less than some specified threshold the samples are grouped together into a cluster. This algorithm is quite versatile as it needs no specification of the expected number of clusters, and can discern spherical and nonspherical clusters as well as linearly nonseparable clusters and clusters with unequal populations. Since this explanation is broad at best the reader should reference "Disaggregative Clustering Using The Concept of Mutual Nearest Neighborhood" by K. Gowda and G. Krishna from "IEEE Transactions on Systems, Man, and Cybernetics", Volume SMC-8, No. 12, December 1978 for a more detailed description.

This discussion of statistical pattern recognition has focused on measurement and classification processes, areas where researchers continuously seek more accurate and efficient algorithms. Classification schemes which may be trivial for a human observer may prove nearly impossible for the computer. For example locating a military base is easily accomplished by a photo interpreter. The observer focuses on pattern of buildings versus the surrounding area. For the computer, however, no true concept of "contextual clues" exists. Thus the computer may focus on lengths of edges and orientation of the image. and the computer may still have difficulties since many characteristics of military bases such as roads and housing are similar to civilian settlements. Yet the computer is often able to make distinctions very difficult for humans, such as fine changes in texture. The thrust of research then should be to incorporate those contextual clues which humans use to perform classifications into schemes plausible for machines. In this way we can take advantage of the positive aspects of both humans and computers.

5. SYMBOLIC IMAGE PROCESSOR

5.1 INTRODUCTION

This section describes "sip", a Symbolic Image Processor. sip is a high level interactive system designed to process images symbolically. It is primarily intended to be a test-bed for new pattern recognition algorithms of a heuristic rather than a statistic nature. As such, sip is an attempt to apply the results of Artificial Intelligence research to image exploitation.

sip has been written in the LISP programming language. The LISP dialect used is the University of Maryland's implementation of the University of Wisconsin Univac 1100 LISP. It is not necessary to know LISP in order to use sip. It is very helpful, nonetheless, to be somewhat familiar with it. This is especially true in writing production rules, for which it is suggested that one understand LISP in some detail. Production rules are covered in greater depth below.

Because sip is an experimental system and because it is written in LISP, new users may find it difficult to get started. To alleviate this problem, an effort has been made to make sip a hospitable environment in which to process images. Every sip command is listed in a menu, accessible from the shell via "menu symb". Manuals for sip are available from the shell and from sip via the "man" command. Also, a "help" command is present in sip. Finally, there are this document and the code itself, which is well documented.

5.2 THEORY OF OPERATION

sip, when used to process an image, will in general be the last AFES program to access that image. All of the more traditional AFES statistical pattern recognition should occur before sip is run. The general processing flow is as follows:

1. Enter image

2. Define a classification method

3. Train and classify

4. Examine classification results

5. Edit the classified image

6. Preprocess the image

7. Run sip

We now discuss the last two steps in greater detail.

5.2.1 Preprocessing

Image preprocessing is a prerequisite for symbolic image processing. Preprocessing is needed because of the way in which images are represented symbolically. The symbolic representation of an image is very different from any pixel-based representation. The exact format for symbolic representation is discussed below in the section entitled "IMAGE REPRESENTATION".

Preprocessing is performed on the 11/34 by a program called "sip_preproc" also known as "spp". sip_preproc takes the classified current working image as input. This image is displayed on the DeAnza display. A series of programs; "region_nam", "region_atr", and "region_bnd"; is executed by sip_preproc. They assign names to regions, extract region attributes, and extract boundary information respectively. The results are transferred over the link to the 11/70 where sip can access them. The entire preprocessing sequence takes roughly ten minutes. Less time is required when there are fewer regions in the image. The user should refrain from executing any other

commands that use the link while sip_preproc is running.

5.2.2 SIP

Once an image has been preprocessed at the 11/34 workstation it may be manipulated symbolically on the 11/70 using sip. sip must run on the 11/70 because it is written in LISP and so occupies vast amounts of memory. It would be desirable to run sip from the 11/34 in order to use the display as processing proceeds, however this is not possible with the current link configuration.

There are several actions a user can take while sip is running. He may access, via "enter", the preprocessed image. Information about individual regions, edges or features as well as information about the entire image can be computed and printed. Attributes that can be computed include area, edgelength, average intensity, location, perimeter, and class type. Several adjacency relations are also provided. Regions may be merged together under operator control to define features. Better still, rules can be defined to recognize features; such rules are executable without user intervention. When processing is complete the user can check his results, and save the updated symbolic image via "store". "restore" can be used to retrieve a previously "store"d symbolic image. This procedure can be repeated until the desired features have been satisfactorily recognized.

Thanks to the interactive nature of sip, the effects of any operation can be verified immediately. It may happen that after performing some operation it becomes necessary to undo it. There are two ways to do this. First, if the symbolic image has been "store"d and the stored version is okay, then "restore" may be used to back up. If there is no usable stored version then "enter" may be used to retrieve the original, unmodified symbolic image.

5.3 IMAGE REPRESENTATION

There are two concepts central to symbolic image processing under the AFES that should be elaborated upon before discussing image representation. The first concept will be called a sip object for want of a better term. It is an entity that represents some portion of an image. A sip object can be a region, edge or feature. r4, 119 and island:13 are typical names of objects. sip objects are the fundamental things we refer to and talk about in the symbolic image domain.

The second concept concerns attributes. An attribute is a property associated with an object. Some attributes of objects are "area", "class", "includes" etc. The attributes of an object are stored on a property list maintained by LISP. The sip function "printprops" will display the attributes of a sip object. In general, regions, edges and features will have different sets of attributes. Those attributes that apply to regions are: "area", "center", "class", "intensity", "level", "part_of". Those attributes that apply to edges are: "edgelength", "level", "image". Only a few attributes apply to features. They are: "class", "includes", "part_of". To modify an attribute one can type in LISP:

```
(put 'object-name 'attribute-name 'attribute value)
```

To access an attribute one can either execute the appropriate sip function if such a function exists or else type:

```
(get 'object-name 'attribute-name)
```

It may be possible to compute an attribute for a feature if the attribute is not explicitly defined for that feature. For example, suppose there is a feature called island:13 that includes r8. To find the area of island:13 sip would first look for an area attribute for island:13. Finding none, an

attempt would be made to sum the areas of any included regions. In this case only r8 is included so the area of r8 would be the result.

Up to this point the words region, edge and feature have been used without being precisely defined. We now rectify this situation. A region is defined as a connected set of pixels of the same class.

Each region is surrounded by a boundary composed of edges. An edge is not a pixel nor a set of pixels; if is a set of "spaces between pixels". An edge has one region on each side, each such region is homogeneous with respect to class. No edge can have the same class on both sides. An edge can be a loop, for example an island surrounded by water might have an edge which is a loop.

A feature is a set of one or more regions that have been recognized as constituting something noteworthy. The regions in a feature are not necessarily adjacent; a storage tank farm could consist of several separated storage tanks; a flotilla may consist of several separated boats. Features can be either cultural or topographical in nature. There is a fundamental difference between features and regions in that features must be identified from regions. This can be accomplished either manually using "combine" or automatically using production rules.

There are three data structures present in sip to represent images. One, called "regs", describes regions; another, called "lsegs", describes edges; and the third, called "feats", describes features.

5.3.1 Regs

regs is a data structure that describes regions. It is implemented as a list. There is one entry in regs for each region. Each such entry contains the region name and a set of edges that constitute the region's boundary. The edges are grouped and ordered so that the edge order indicates which edges are

next to one another. For example, suppose we have a doughnut shaped region r4 whose outer boundary consists of l1, l2, l3, and l4; and whose inner boundary consists of l5. Then we would find the following entry in regs:

```
(r4 ( (l1 l2 l3 l4) (l5) ))
```

To retrieve the entry from regs for a particular region, say r4, the following LISP function can be used:

```
(assoc 'r4 regs)
```

To view all of regs just type "regs" (without the quotation marks). r0 is by definition the rest of the world that is not in the image.

5.3.2 Lsegs

lsegs is a data structure that describes edges. It is also implemented as a list. There is one entry for each edge. Each such entry contains the edge name and a list of regions on either side of the edge. For example, suppose l3 separates r4 from r9. Then the following entry might be present in lsegs:

```
(l3 (r4 r9))
```

To retrieve the entry from lsegs for a particular edge, say l3, the following LISP function can be used:

```
(assoc 'l3 lsegs)
```

To view all of lsegs just type "lsegs" (without the quotation marks).

5.3.3 Feats

feats is just like regs. Only the names are changed. Features are stored on a list with their boundaries. feats is updated by the "combine" function or by the "make" function in production rules. To retrieve the entry in regs for a particular feature, say island:13, the following LISP function can be used:

```
(assoc 'island:13 feats)
```

To view all of feats just type "feats" (without the quotation marks).

5.4 COMMANDS

5.4.1 Invoking SIP

sip is invoked from the shell by typing "sip", optionally followed by a list of input filenames. Any such input files are read in by LISP and treated like LISP source code. Production rule files, if present, should be loaded in this fashion. LISP will not complain if a named file is not present, the file name is just ignored.

The sip program itself is only a shell interface. It first executes "pump" to pipe some LISP code to initialize a few important file names. Then LISP is run with $v/sip.cmp as input, along with any other files as explained above. $v/sip.l is the actual source code for symbolic image processing. To look at the code, execute "lkf sip.l". It is liberally commented. $v/sip.cmp is the compiled version of sip.l. Compiling the source code results in a five-fold improvement in speed.

There may be up to a 15 second pause after "sip" is typed before loading is complete. A message will be printed and the user will find himself communicating with the LISP interpreter.

The LISP interpreter signals that it is ready by printing "Eval:". Any sip command (or other LISP function) may now be typed. The command will be evaluated and the resultant value displayed before control returns to the LISP interpreter. This read-eval-print cycle continues until one exits LISP.

5.4.2 LISP Commands

The LISP interpreter will attempt to evaluate every expression given it. If it is given a single variable name, called an atom in LISP, it will print out the contents of that atom. This is useful for viewing regs, lsegs, feats, classlist, or rulelist. Typing the name of an atom that has not yet been defined is an error.

To execute a LISP function, one must enclose the function name and arguments in parentheses. Since sip command are just LISP functions this is the way sip commands get executed.

Some functions require that their arguments be quoted. Quoting an object consists of prefixing it with a single quote. Most, but not all, sip functions require that their arguments be quoted. Failing to use a quote where it is needed is a significant cause of LISP errors.

5.4.3 LISP Errors

One kind of error, leaving out a quote, was mentioned above. When it occurs LISP will usually type:

```
WARNING, atom-name IS UNBOUND
Help:
```

At this point the LISP interpreter is confused and is asking for help. The simplest cure is to send an interrupt by hitting the rubout key. This tells the interpreter to forget whatever it is doing and get back to "Eval:"

mode.

Another frequent error is the misspelling of atom or function names. The message is the same as above. So is the cure.

Forgetting the parentheses around a function is not an error as far as LISP is concerned. The interpreter treats the function name like an atom and prints the LISP code for that function. This is usually not what you want. As long as LISP prompts with "Eval:" no special action needs to be taken to get out of trouble.

There are many other possible LISP error modes, most of them characterized by the "Help:" prompt. The safest thing to do is to send an interrupt. An interrupt will always get LISP back together no matter what the interpreter was doing.

5.4.4 LISP Quirks

Our version of LISP is somewhat non-standard. Here are the more important anomalies:

case — The LISP interpreter ignores upper and lower case distinctions. It is all converted to lower case internally. The one exception is within double quotes, but this does not concern us here. Upper and lower case characters can be intermixed within atom names without ill effect. The judicious choice of case may make LISP code more readable. See $v/sip.l for example.

EOF — The end of file character, also known as EOF or control-d, is used to exit from LISP in response to "Eval:". EOF in response to "Help:" will normally get you back to "Eval:" mode. Thus, two EOF's will always suffice to get out of LISP. "(return)" in response to "Eval:" will also cause LISP to exit.

Comments The comment character is a question mark. All text on a line to the right of the comment character is ignored.

() The LISP interpreter will not evaluate an expression until the parentheses balance. So one can type an expression on more than one line; evaluation will not proceed until the leftmost parenthesis has been matched. Square or angle brackets can be used in place of parentheses. As a bonus, a special convention allows one to do some automatic parenthesis balancing. The rule is: any closing bracket;),], or >; will generate the correct closing brackets to close all opening brackets; (, [, or <; to the left back through the opening bracket corresponding to the closing bracket that was written. For example, writing a] will force closure of all lists starting with (or < back to the left until an unclosed [is found to close.

Real In order to distinguish real numbers from dotted pairs LISP requires that real numbers be prefixed by a percent sign. Real numbers are preceeded by a percent sign on output, and must be preceded by one when input. Examples of real numbers are: %1.0, %-.57, %357.110e-12. This is a kludge that hopefully will be fixed in updated versions of the interpreter.

Loading When loading LISP source code files, the interpreter does not generate any error messages. Therefore it is possible that a file you thought was loaded has been ignored. In the case of production rule files this can be checked by typing "rulelist" to see if the rules were defined. To load a file while LISP is running, the user can type:

```
(load "file-name")
```

Shell There is a special Unix LISP command, "(sh)", that enables one to temporarily suspend execution of the LISP interpreter and begin execution of a shell. Any shell command can then be issued. An EOF (control-d) from the shell will return the user to LISP.

5.4.5 Sip Commands

A list of sip commands is available from the shell by typing "menu symb". A more detailed description can be had by typing "man sip afes". For even more detail about individual commands there is an on-line afes manual for each. There are two sip commands that do not have on-line documentation, the help and man commands. The sip help command, as opposed to the afes help command, can be run in sip by typing "(help)". It prints a list of all available sip commands and their syntax. The sip man command, as opposed to the afes man command, can be run is sip by typing "(man manual-name)". This will cause the execution of the afes command "man manual-name afes" from sip by forking a shell. This is especially useful for reading on-line manuals for sip commands while sip is running. For these reasons this document will not describe the commands further, with the exception of production rules. Suffice it to say that sip commands conform in every way to LISP syntax.

5.5 PRODUCTION RULES

Production rules are a means of expressing domain dependent knowledge in declarative form. Any production rule system consists of three components; a set of rules, a data base, and a rule interpreter. In the sip system the ruleset can be stored in a file or typed into sip directly. The database consists of the symbolic image as stored in memory and the rule interpreter is just the LISP interpreter.

Rules have four parts. The first is the word "rule". This tells the production rule system to define a rule. Next is a name to assign to the rule. The name is arbitrary and can have any desired length. Then comes a

set of conditions which, if true, will cause the specified actions to be performed. A typical rule might be:

```
(RULE FINDPARKS
        (IF     (CLASS_IS (trees tree grass veg))
                (AREA_IS  (BETWEEN 200 5000))
                (SURROUNDED_BY (resid urban) (GREATERP %.90))
        )
        (THEN   ()
                (MAKE park)
        )
)
```

Here, key words are capitalized for clarity, since LISP ignores case. Typing this code in or loading it from a file will only define a rule, not cause its execution. To execute a single rule the user should type in the rulename surrounded by parentheses. As the rule executes, each region will be examined to see if the condition part of the rule is true. If it is then the specified action is taken before the next region is examined. If the condition is false then no action is taken before the next region is examined. It is possible to execute all rules at once by typing:

```
(mapc rulelist eval)
```

Basic documentation on the clauses used for production rules is available via "man rule afes". This manual should be read before continuing with this section. Examples of production rules are available via "lkf rule_exam". The following subsection describes the internal workings of production rules.

5.5.1 Rule Declaration

When a production rule is defined, the rule macro is executed to set up the rule. The rule name is first added to rulelist, the list of defined rules. Then the following code is created and executed:

```
(CSETQ rulename
        (LAMBDA ()
                (MAPC regs
                        (LAMBDA (region)
                                (SETQ region (CAR region))
                                (COND   (conditionpart actionpart))
)       )       )       )
```

Thus when rulename, a function of no arguments, is executed, each region will be checked to see if the condition is true. If so, then evaluate actionpart. Each region is checked in this manner.

Note that the local variable "region" holds the region name. The production rule clauses area_is, class_is, perim_is, surrounded_by, and make all use "region" implicitly. A knowledgeable production rule user (defined as anybody who has read this far) can add his own clauses using "region". For example, a rule to print the names and classes of all regions whose class is urban or industrial could be written:

```
(rule urb_ind
        (if     (class_is (urban industrial)))
        (then   ()
                (print region)
                (print (class region))
)       )
```

To change the class type of all shadows from shadow to unknown we might use:

```
(rule shad_unk
        (if     (class_is (shadow)))
        (then   ()
                (put region 'class 'unknown)
)       )
```

Finally, here is a rule to recognize rivers (which I will arbitrarily say have an area/perimeter ratio less than 5).

```
(rule findriver
        (if     (class_is (water))
                (lessp (quotient (area region) (outerperim region)) 5)
        )
        (then   ()
                (make river)
)       )
```

Production rules should be kept in a file and loaded when sip is invoked. In this way the rules can be easily edited with your favorite editor. Of course, you may type them into LISP directly if you prefer.

5.6 SAMPLE DIALOG

```
% sip rules                              *** user types "sip rules"
ULISP V1.7   Copyright 1978,R.L.Kirby
Eval: SYMBOLIC IMAGE PROCESSOR UP AND RUNNING
Value: t
Eval: (enter)                           *** user types "(enter)"
LINES READ: 93
REGIONS READ: 47
```

```
CLASSES READ: 4
Value: t
Eval: regs                                  *** user types "regs"
Value: ((r0 ((11))) (r1 ((11) (166 150 137 129 121 18 14 13 123 122 125 17
 16 115 136 134 133 127 126 132 143 140 139 142 168
 169 170 164 159 160 161 163 165 172) (178) (181) (187) (188) (189) (190)
 (191) (192) (193))) (r3 ((166 151 137 130 121 19 14
 12 123 124 136 135 133 128 126 131 143 144 141 142 167 169 171 164 158 154
 155 156 161 162 165 173) (147 146 149 152) (174) (175)
 (176) (177) (179) (180) (182) (183) (184) (185) (186))) (r4 ((12 13))) (r2
 ((115 15 17 119 113 110 112 116 118 120 122 124)))
 (r8 ((15 16))) (r7 ((18 19))) (r10 ((120 117 116 111 110 114 119 125))) (r9
 ((111 112))) (r11 ((113 114))) (r12 ((117 118)))
 (r14 ((127 128))) (r16 ((129 130))) (r17 ((131 132))) (r19 ((134 135))) (r21
 ((140 138 144))) (r23 ((139 138 141))) (r27 ((145
 147))) (r29 ((148 146 145 152))) (r30 ((148 149))) (r28 ((150 151))) (r32
 ((153 155))) (r33 ((157 154 153 156 160))) (r35 ((158
 157 159))) (r36 ((162 163))) (r43 ((167 168))) (r39 ((170 171))) (r37 ((172
 173))) (r5 ((174))) (r6 ((175))) (r13 ((176))) (r15
 ((177))) (r18 ((178))) (r20 ((179))) (r22 ((180))) (r24 ((181))) (r25 ((182)))
 (r26 ((183))) (r31 ((184))) (r34 ((185))) (r38
 ((186))) (r40 ((187))) (r41 ((188))) (r42 ((189))) (r44 ((190))) (r45 ((191)))
 (r46 ((192))) (r47 ((193))))
Eval: (area 'r4)                            *** user types "(area 'r4)"
Value: 7615
Eval: (neighbors 'r4)                       *** user types "(neighbors 'r4)"
Value: (r3 r1)
Eval: rulelist                              *** user types "rulelist"
Value: ((findtanks) (findbridges) (findboats) (findislands) (findparks))
Eval: (findislands)                         *** user types "(findislands)"
island:1                                    *** sip finds an island
Eval: feats                                 *** user types "feats"
((island:1 ((178))))                        *** compare this with r18 above
```

```
Eval: (area 'island:1)                    *** user types "(area 'island:1)"
Value: 8718
Eval: (printprops 'island:1)              *** user types "(printprops 'island:1)"
island:1
((includes r18) (class . island))
Value: nil
Eval: (printprops 'r18)                   *** user types "(printprops 'r18)"
r18
((part_of island:1) (level . 2) (intensity . 40) (center 216 . 131)
 (area . 8718) (class . veg))             *** notice part_of and area values
Value: nil
Eval: (EOF)                               *** user types control-D
%
```

5.7 LIMITATIONS

5.7.1 Representation Limitations

There are some shortcomings to the current representation scheme. Possibly the most serious is that regions are segmented based upon class type. In order to recognize a feature, its component regions must be segmented. For example, to find roads in a city requires that the road be of a different class from its surroundings. But if roads, houses and factories are all classified as "urban", it will not be possible to recognize the road alone. The solution to this problem seems to lie in the development of classification methods that can distinguish roads from other urban areas.

There are some attributes of regions that one would like to have around but are difficult to measure or represent. Shape is one such attribute. It would be nice to be able to declare in a production rule that storage tanks are round, trucks are rectangular, etc. We do not now have this capability. Although there several sip functions that deal with adjacency, none deal with the proximity of one region to another in a more general sense. Fuzzy

concepts like this are difficult to express in any programming language.

5.7.2 Rule Limitations

As currently implemented, production rules can only take one region at a time and make a feature of it. In order to combine more than one region into a feature it is necessary to manually use the "combine" sip command. This should be remediable with some more work. It might also be desirable to let production rules operate on edges and features as well as regions. Perhaps each rule should specify whether it is to look at regs, lsegs, feats or some combination thereof. One last drawback of rules is that if the same rule is run more than once on the same image, it will recognize the same features each time, giving several names to the same actual feature. The "make" function should be fixed to prevent this.

5.7.3 Number of Regions

Running as it does on a PDP-11, sip has storage limitations. There is an upper limit of roughly 200 regions that can be fit into memory at once depending upon the boundary configuration. This is actually not as many regions as it may seem, so some editing will be required on most images before sip_preproc is run. im_edit can be used to manually reduce the region count, and rm_noise can be used to automatically reduce the region count. If there are too many regions, LISP will print a nasty message when "(enter)" is typed. To process more regions we could use a larger machine, such as a VAX.

5.7.4 Speed

The preprocessor, which takes roughly 10 minutes of PDP-11/34 time, is one bottleneck, but its performance is not readily improved upon. sip itself uses compiled code, and so runs fairly quickly. Typical times are 15 seconds to get sip running, 20 seconds to "enter", and 5 seconds to "restore".

6. AFES IMAGE PROCESSING LANGUAGE

6.1 INTRODUCTION

The purpose of the AFES Image Processing Language (IPL) is to provide the experimentor with a language which he or she can use to specify the AFES tools and parameters for an image processing experiment, and to set up a sequence of experiments to be run automatically in a "batch" mode. Such a sequence of experiments could involve application of a series of different processes to a single image, repeated alteration of the functional parameters of a single process, application of a single process/parameter set to a sequence of different images, or any combination thereof. For example, one might wish to apply a number of different edge enhancement operators (e.g. Sobel, Roberts, Cross, Laplacian, etc.) to an image then assess the results of each one. Or a pixel classification procedure might be applied repeatedly to an image, with each iteration using a different combination of measurement extractors or different values of measurement extractor parameters, e.g. window sizes, thresholds, weighting factors, etc.

The Image Processing Language consists of a table structure which specifies how an experiment is to be conducted, a set of special commands for modifying the table, and a control structure which supervises the definition and application of the experimental procedure. Each user has one or more private versions of each of these structures.

6.2 TABLE STRUCTURES

The first structure specifies what image is to be processed. The AFES identifies an image according to photo, view, and frame. The photo is the basic unit of source material, such as a single film clip. This is divided into views, with one view being the size of a typical AFES image, e.g. 1024 x 1024 pixels. The view is further divided into frames where a frame may be the

left or right conjugate of a stereo pair or simply a single (monoscopic) image.

The second basic table structure for the IPL is the AFES "method file". It constitutes of a "recipe" for an image processing experiment. A typical method file for statistical pattern recognition as applied to individual pixels might be:

```
measurements:
      avg 3
      lapl

classifier:
      mahal  [optional arguments]
(training set)

class:
trees
      regions:
            trees1
            trees2

class:
water
      regions:
            lake1
            river1
            river2

class:
urban
      regions:
            industry1
            residential1

comments:
    <user comments>
```

The measurements are the AFES pixel measurements to be extracted from each image pixel. A list of available measurements is contained in the meas menu.

The particular classifier to be used is the next entry. A variety of supervised and unsupervised classifiers are listed in the class menu. In the above example, the Mahalanobian classifier is used, which requires supervised training. Hence, a specification of the training regions to be used for each desired output class is given. Each user has a collection of training regions which are defined interactively via the get_region command.

Initial versions of AFES support method files for pixel classification; future extensions are anticipated which will incorporate similar types of specifications for edge-based region extraction (as opposed to pixel-based extraction), region classification, and symbolic processing.

6.3 IPL COMMANDS

The image processing language uses a set of special commands which perform non-interactive modification of the table described in the preceding section and apply procedures defined in the method file to the image defined by the current photo, view, and frame.

6.3.1 Change Processing Image (cpi)

The purpose of cpi is to change the image being processed from within the image processing program. Typical uses would involve applying the same processing method to a number of different images.

The syntax is:

cpi <photo> <view> <frame>

6.3.2 Change Processing Method (cpm)

The cpm command allows the IPL program to change from one processing method to another. A user would typically have defined a collection of methods, each given a unique name or "method-id". The syntax for changing the

method from within an IPL program is:

cpm <method-id>

6.3.3 Modify Method (mod_method)

The mod-method command is the most powerful special function which the IPL uses. It allows the user to alter the processing method from within a program. A number of flags are used to specify how the method is to be changed.

The command syntax is:

```
mod_method(mdm)   [<-ma "add_measure" ["del_measure"]> ||
                  <-md "measure"> || <-c classifier> ||
                  <-ra class region> || <-rd class [region]> ||
                  <-o "comments"> || <-sv method_id>    <-ls>] ||
```

where, as usual, <arg> denotes a mandatory argument and [arg] an optional argument.

The following is a description of the flags and how they work:

- ma -the first argument string in quotes is a measurement string to be added. The second argument string in quotes is optional and represents an old measure to be deleted from the method file. When the second argument is present the function is a replace measurement.

- md -delete the measurement string in quotes

- c -changes the classifier name and/or any optional arguments

- ra -adds the region to the class and adds the class if needed

- rd -deletes region from class and entire class if the optional region name is not given

- o -adds comment line in quotes to comments section

- sv -saves the current method including classified output under a new method name

- ls -lists the contents of the method file to standard output

The "-sv" switch would be especially useful if a method works very well and the user would like to save it but continue experimenting with the current method.

6.3.4 Current Methods and Images

A user performing image processing experiments may run the required programs in the foreground. That is, when a program is started, input from the terminal is suspended until the program finishes and returns control to the terminal. IPL programs executing in foreground operate on and use the AFES current "working image" and "working method".

UNIX also provides the capability to initiate "background" processes, i.e., input from the terminal continues while the background process executes. IPL programs can run in the background and in fact it is often convenient for them to do so, so the user can continue with program development, foreground experiments, etc., while a lengthy IPL program executes in the background. In this situation, AFES provides a separate current image and method for each background process. These are referred to as the "processing image" and "processing method" to distinguish them from the "working" image and method associated with foreground processes. The "processing" image and method are automatically invoked by cpm and cpi commands which are executed by an IPL

program in background. The important things to remember are that (1) an IPL program running in the foreground always operates on and uses the current "working" image and method, and that (2) a background IPL program will also operate on and use the current working image and method unless it contains explicit cpm and cpi commands to change to a current "processing" image.

6.4 CONTROL STRUCTURE

The preceding sections describe the tables which define how an image is to be processed and the commands which allow modification of the table by a background process, i.e., an IPL program. The last component is the control structure which ties these commands together into a stand-alone program. This structure is the shell, the UNIX command language. This paper will not attempt to provide a full description of the shell; rather, its use to generate IPL programs will be described. Readers unfamiliar with the shell should read the PWB/UNIX Shell Tutorial by J.R. Mashey, published by Bell Telephone Laboratories and included in the UNIX system documentation.

An IPL program consists of a sequence of shell commands and IPL special commands which reside in a file identified by some program name. After creating a program file, it must be marked as executable using the "chmod" command. Up to nine arguments can be passed to a shell program; these are referenced in the program itself as a character sequence of the form $n where n is 0 to 9. $0 is the name of the program, while $1 to $9 refer to the arguments which follow the program name when it is executed. The following examples are taken from the topical document entitled "AFES System Structure."

```
:     'tnis file runs the method "test 1" on
:     'the photo: syracuse
:     'for these views: view1, view2, view3
:     'and this frame: mono

set a = 1
cpm test1
while $a ! = 4
cpi syracuse view"$a" mono
cfy
set a + 1
end

:     'this file runs the methods "test1-test3" on
:     'the photo syracuse center mono

set a + 1
cpi syracuse center mono
while $a != 4
cfy
set a + 1
end

:     'For the photo -syracuse center mono
:     'this file starts with method - test1
:     'trains the classifier for the method
:     'makes a copy of test1 named test2
:     'makes test2 the current processing method
:     'deletes class "trees" from test2
:     'adds class "green" with region "field"
:     'The confusion matrices may be examined later

cpi syracuse center mono
cpm test1
train
mdm -sv test2
cpm test2
mdm -rd trees
mdm -ra green fields
mdm -ma "contr 6 / avg 20"
train
```

Any parameter of the method files may be modified by the "mdm" command and any parameter of the "cpi" or "cpm" commands may be modified. The user may examine the output visually as with "map" for a classifier output or tabularly as with the confusion matrix for the train command.

7. PHOTOGRAMMETRIC SOFTWARE

7.1 INTRODUCTION

This section describes the photogrammetric software available on the Automatic Feature Extraction System. It is intended to provide insight into the design of the software as well as give guidance to the mathematical conversions followed during implementation.

The AFES is designed for experimentation with digital imagery. However, the Scanner/Viewer Subsystem provides for hardcopy as one of the primary inputs to the testbed. As a result the topic of all digital photogrammetric techniques is addressed in light of hardcopy sources. Some aspects of a truly all digital system are mentioned but are not the main topic of this report. Certainly many of these techniques discussed are applicable to the all digital environment.

Section 7.2 begins with a brief summary of the goals of the AFES design. Subsections within 7.2 discuss specific requirements which are met by photogrammetric processes, including maintaining stereo and performing mensuration and point positioning, Section 7.3 which follows, provides a detailed description of the AFES mensuration package.

7.2 THE AFES SYSTEM DESCRIPTION

The Automatic Feature Extraction System (AFES) is an integrated hardware/software complex. It is designed as a test bed for applying image processing, photogrammetry, pattern recognition, and artificial intelligence derived techniques to Defense Mapping Agency (DMA) requirements for semi-automatic map generation and updating. The AFES has been designed as a complete man-machine system for image understanding, and an efficient receiver of algorithms. AFES possesses facilities for easily reimplementing,

integrating and testing algorithms developed elsewhere, as well as new algorithms. The system contains elaborate facilities for image (input) and storage, and can be operated by persons unfamiliar with computers.

Film and map inputs to the system are processed by the viewer/scanner and plotter/scanner, respectively. The viewer/scanner is capable of scanning a 1024 x 1024 pixel image with 256 grey levels in approximately 17 seconds. The image size on the film plane is variable from 5 mm square to 30 mm square with arbitrary rotation and skew up to 20 degrees. The viewer/scanner is photometrically calibrated and geometrically accurate. The map plotter/scanner accommodates both opaque and transparent input with scale and rotation fixed. The output from the plotter/scanner is a high quality manuscript.

7.2.1 System Requirements

There are a variety of interactive, interpretive functions that require photogrammetric processing. The essential problem is to take measurements on the displayed imagery and produce coordinate outputs in a ground reference system. In addition, the analyst interprets the photos in stereo for most tasks in his normal working environment. The automatic maintenance of the stereo model is an essential function of the AFES test bed.

The AFES experimental functions require three basic categories of photogrammetric processing the maintenance of a stereo model on imagery being scanned, object mensuration, and point positioning. The system was not required to perform as a stereo compiler with real time rectification. Instead the user selects subimages to work on in stereo. Design requirements did not call for constant real time mapping of object space to image space. It was envisioned that discrete points would be measured as opposed to functions requiring constant point collection such as profiling. Therefore, care was taken to allow the greatest flexibility in selecting scenes to be scanned, and positioning the cursor for point selection. The system operates

as an image space plotter with a rectified duplex of stereo cursors on the screen for the operator to measure. The rest of the section will discuss each of the three basic processes that AFES is designed to perform.

7.2.2 Maintaining Stereo

The task of maintaining the stereo model is accomplished by keeping the epipolar lines parallel to the viewer's eye base. This condition is maintained through different mathematical computations for different sensors. The scale of each image must also be adjusted so that each is displayed at the same scale. Finally, a cursor must be displayable in stereo for manual delineation of features in the imagery. Since the primary image input is stereo hardcopy images from the scanner, the main body of code for stereo maintenance is designed for control of the scanner. There are routines, however, for rectifying two conjugate images for stereo viewing if their source is other than the scanner.

The maintenance of stereo requires knowledge of the orientation information which is output from a triangulation or block adjustment procedure. AFES relies upon the output of other photogrammetric systems to provide the appropriate attitude and position data for images being processed. There are no orientation adjustment routines provided on the AFES for this purpose.

7.2.3 Mensuration

Some of the ground space measurements that are obtained from the image data are relative measures. That is, the height of a tower or length of an object are often required. This mensuration process can be performed by local image measures of shadow or local differential parallax. In the AFES however, there is enough information available that rigorous models can be used even for mensuration. The mensuration programs operate in a local vertical space as does the point positioning software.

The routines available for mensuration are interactive. They allow the operator to measure height changes and planimetric distances. Incorporated into the software that manipulates the region data generated by the Statistical Pattern Recognition Modules are measures that relate areas of contiguous regions to areas on the ground. These routines are based on local scale derived by height information and the sensor models.

7.2.4 Point Positioning

There are two methods of point positioning which would be applicable to the AFES. The first is point positioning with rigorous sensor model. This could be performed stereoscopically or monoscopically with elevation data included. Alternatively, a warping procedure can be used to fit a single image to a ground coordinate system. Unfortunately, this warping procedure does not incorporate any relief distortion. However, in areas of relatively flat terrain a warping procedure would be adequate and the algorithms are very fast.

The AFES uses both rigorous models and warping for point positioning. The rigorous models are incorporated in a series of interactive routines which allow the operator to select in stereo or monoscopically the points which he wishes to measure. The system supports several sensor models and provides outputs in a variety of ground coordinate reference systems. The warping software accepts control point data from the operator while the operator measures the points on the sub-images displayed. The corresponding point values are then used to generate the geometric transformation from image to ground. These geometric transformations, based on two dimensional polynomial transforms, are then utilized by the resampling routines to generate an image registered to the ground reference grid of the central points. The transforms for registration include first through fifth order polynomials.

7.3 MENSURATION PACKAGE

The AFES "mensurate" command allows the user to perform mensuration tasks interactively. Measurements for these commands are performed on the specified display channel[s], which are specified as an argument to 'mensurate'.

- -s : use the left and right monochrome display channels
- -c : use the red and green color display channels (anaglyph)
- -R : use the red channel only
- -G : use the green channel only
- -B : use the blue channel only
- -l : use the left channel only
- -r : use the right channel only -p : if present as a second argument specifies current header information data is present in mensuration form that matches what is currently on the display

Upon invocation, after a short delay for preprocessing the header information, the prompt 'enter command > ' will appear on the terminal. If the return key is typed in response to this prompt a menu will be printed out on the terminal. The prompt, for entering a command after the initial prompt, will be a '> '. A menu of available commands can also be found during execution by typing 'commands'.

Available mensuration commands include:

spts : Stereo point positions

relpts : Finds relative point positions

height : height of objects

distance : xy distance between points

phopts : Find photo point positions (stereo)

mpts : single photo local xy points

mphopts : single image photo points

quit : End execution of program (or stop)

save : save output in a file

commands : List available commands

sh : Escape temporarily to another shell

geographic : output in geographic coordinates

geocentric : output in geocentric coordinates

lambert : output in lambert coordinates

local : output in local xyz

mercator : output in mercator coordinates

polar : output in polar coordinates

utm : output in utm coordinates

More explicit usage for each command is given below.

If 'mpts' is the command selected then a cursor will appear on the chosen display. An optional argument to this command is 'left' or 'right'. This can be abbreviated by using an 'l' or 'r'. The optional argument is only needed if operating in stereo mode and the right display is to be selected. (The left display is the default display in stereo mode). If operating in anaglyph stereo consider the red image channel to be the left display and the green display as the right display. The left trackball button selects a point. The right trackball button toggles the cursor from crosshair to dot. The center button has no effect. After a point has been chosen the user enters the elevation of the point on the terminal. The local xyz ground coordinates are then printed on the terminal.

The command 'mphopts' allows the user to select photo points for a single image channel. An optional argument to this command is 'left' or 'right'. This can be abbreviated by using a 'l' or 'r'. The optional argument is only needed if operating in stereo mode and the right display is to be selected. A cursor will be displayed on the center of the selected or default display. There are two modes of operation for the trackball while in 'mphopts'.

I. If the center button (red button) is up, cursor movement will be controlled by the trackball. The left trackball button when depressed will select an image point for transformation to the photo system. The resultant photo point will be written to standard output (the terminal is the default standard output). The right button on the trackball when pressed toggles the cursor from a crosshair to a dot, or a dot to a crosshair.

II. If the center button is down, the trackball will be in a zoom and scroll mode. In this mode, movement of the trackball will scroll (hardware scroll) in x and y with the trackball motion. The left button when pressed increases the zoom factor. The right button decreases the zoom factor. A 1X zoom factor is the initial state. Possible zoom factors are 1X, 2X, 4X and 8X.

The command 'mphopts' terminates when the same point on the image is selected twice in succession.

The commands 'relpts', 'height', 'distance', 'phopts', 'spts' all use the same trackball operation. A description of the functions of the trackball and its buttons will be given here. If -s was specified as an argument to mensurate(msr) the left and right monochrome displays are used. There is a trackball for each display. The left trackball has the following functions when a command is being executed.

The left button selects a point on the left and right displays beneath the cursors positions.

The right button on the left trackball changes the cursors type. If the cursors were crosshairs and the button is hit they are changed to dots. If the cursors were dots they are changed to crosshair.

The center button on the left trackball controls what happens when the left trackball is moved. If the button is in the up position, trackball movement will control movement of both cursors. If the button is down the cursors will remain fixed and the left and right image will be scrolled.

The right trackball has the following functions:

The left button on the right trackball allows for an increase in the zoom of the displays when hit. The maximum zoom is 8X. The zoom is centered around the position of the cursor on the left display.

The right button on the right trackball allows for a decrease in the zoom of the displays when hit. The minimum zoom is 1X. The unzoom is centered around the position of the cursor on the left display.

The center button on the right trackball determines what happens when the right trackball is moved. If the button is up then movement of the trackball will move the cursor on the right display in the x direction only. If the button is down then movement of the right trackball will move the cursor on the right display in the y direction only.

If the -c is used then the red and green image planes are used in the stereo mode. A single cursor appears on the display and a point is selected when the cursor, the desired point on the red image and the corresponding point green images appear at the same position. The trackball has the following functions to enable image point selection.

The left button always selects a point when pressed.

The right button changes the cursor from a crosshair to a dot or a dot to a crosshair. (There is one exception explained below)

The center button changes the mode of operation for trackball movement. There are 3 modes of operation for the trackball. Annotation memory for the display (characters written on the display) show the current mode.

move cursor - allows movement of the cursor with the trackball

scroll red in x - allows the red image to be scrolled in x only. The general procedure is to use this mode to correct x parallax. The cursor can be 'floated' in this mode and positioned on a point.

scroll green - allow the green image to be scrolled in both x and y. Also in this mode the right hand trackball button controls a zoom mode. Pressing the right hand button will zoom from 1X to 2X to 4X to 8X and back to 1X while in this mode.

The command 'height' is used to find difference in elevation of two points. Two point pairs are selected on the images with the cursors. The height is output on the terminal. If the value returned is positive the first point had a higher elevation. To terminate the height command select the same point consecutively.

The command 'distance' is used to find the xy distance between two points. Two point pairs are selected on the images with the cursors. The distance between the two points are output on the terminal. To terminate this command select the same point consecutively.

The command 'relpts' is used to give relative point positions. The output system is a local vertical system. Two point pairs are selected on the images. The local xyz ground coordinates and the residual parallax are returned after every point chosen. After the second point is chosen for each set the difference in x,y and z are printed and the distance between the points in xyz. To terminate this command select the same point consecutively.

The command 'spts' is used to compute point positions in one of several available coordinate systems. The default system is local vertical. The output system can be changed to a different projection by typing the name of the map projection desired while in a command mode for 'mensurate' (a command mode is when a '>' appears as a prompt). Available output systems are: geographic, geocentric, lambert, mercator, polar, utm. Parameters for map

projections (such as standard parallels or central meridean) should be entered by using the command 'geo' on the workstation before 'mensurate' is executed.

Besides having output printed on the terminal it can be saved in a file as well. To do this just type 'save' while in a command mode (when '>' is the prompt) for 'mensurate'. A message will come back asking the name of a file where you want the output to go. If the file already existed the output will be appended to the end of the file. If the file did not exist it will create and write to that file. If a carriage return is hit for the response the commands will no longer write to this file. All commands that produce terminal output will write to the 'save' file if one exist. The command 'save' must be typed any time 'mensurate' is run if there is to be saved output. The command 'stop' or 'q' will stop end execution of 'mensurate'. The command 'sh' will fork a new shell.

8. SCANNER SUBSYSTEM

8.1 INTRODUCTION

The AFES Scanner System provides the means by which stereo photographs and graphic input materials are scanned and digitized for use in the AFES system. The scanner system is a separate computer controlled system which incorporates linear array charge coupled device (CCD) image sensors in conjunction with computer controlled translation stages to perform the scanning functions required by the AFES System. Scanner operation is essentially independent of the rest of the AFES System, and it performs scanning and digitizing operations simultaneously with other AFES system operations.

The scanner system consists of a stereo photograph scanner/viewer unit, a graphics scanner/XY plotter, a control computer, special real-time digital video processing hardware, and a control interface. The photograph scanner/viewer has two photograph stages equipped for both scanning and viewing. The use of two stages permits selected areas of either photograph of an oriented stereo pair to be scanned and digitized on demand. Input map base material and other graphics are scanned with the graphics scanner which also serves as an XY plotter for output graphics plotting. The control computer is the central control element of the system. It controls the scanner servos and video processor to implement scanning and digitizing functions. It also controls the flow of data between the scanner and the rest of the AFES System, and provides the means for operator communication with the scanner system. The following subsections provide a more detailed description of each component of the scanner.

8.2 AFES SCANNER SYSTEM

A diagram of the AFES scanner system is shown in Figure 8-1. Major subsystems include the two photograph scanners, the graphic scanner, video processing and sensor control hardware, the control interface and a Digital Equipment Corporation PDP 11/34 system control computer. For operator interaction during setup, a control panel and CRT terminal are also included. The main interface between the digitizing system and the AFES system computer (PDP 11/70) is through one port of a dual access mass storage disk unit. The primary function of this interface is to permit transfer of image data from the scanner system to mass storage without interfering with other AFES system operations. A PAR-developed contiguous file allocation algorithm has been implemented for this dual-ported disk, which permits fast access to image files by the scanner, which must write to contiguous disk blocks. At the same time, these images can be accessed as ordinary UNIX files on the PDP-11/70 side.

The two photograph scanners consist of servo-driven XY photo stages, servo-driven zoom and rotation optics and solid state linear array photosensor subsystems. Photograph scanning is accomplished by moving the stage at a constant velocity in one direction and sampling lines of imagery along another direction with the array sensor. Since the array is read out sequentially, this procedure results in a raster-type scanning of the photograph, similar to that produced by a TV camera but at a slower rate. The output of the scanner is a video signal which is transmitted to the video processing hardware. The computer-controlled zoom allows for scanning the photographs at a wide range of resolution. Minimum pixel size is about 5 microns and maximum pixel size is about 30 microns. The minimum pixel size was defined based on informal studies of various spot sizes conducted by PAR and government personnel. The range of 5-30 microns stems from the capability of the optical zoom system. Actual maximum resolution as measured during acceptance testing is approximately 80 lp/mm. The scanner can produce image data in a raster format which is approximately 1000 pixels wide and of arbitrary length. Normally,

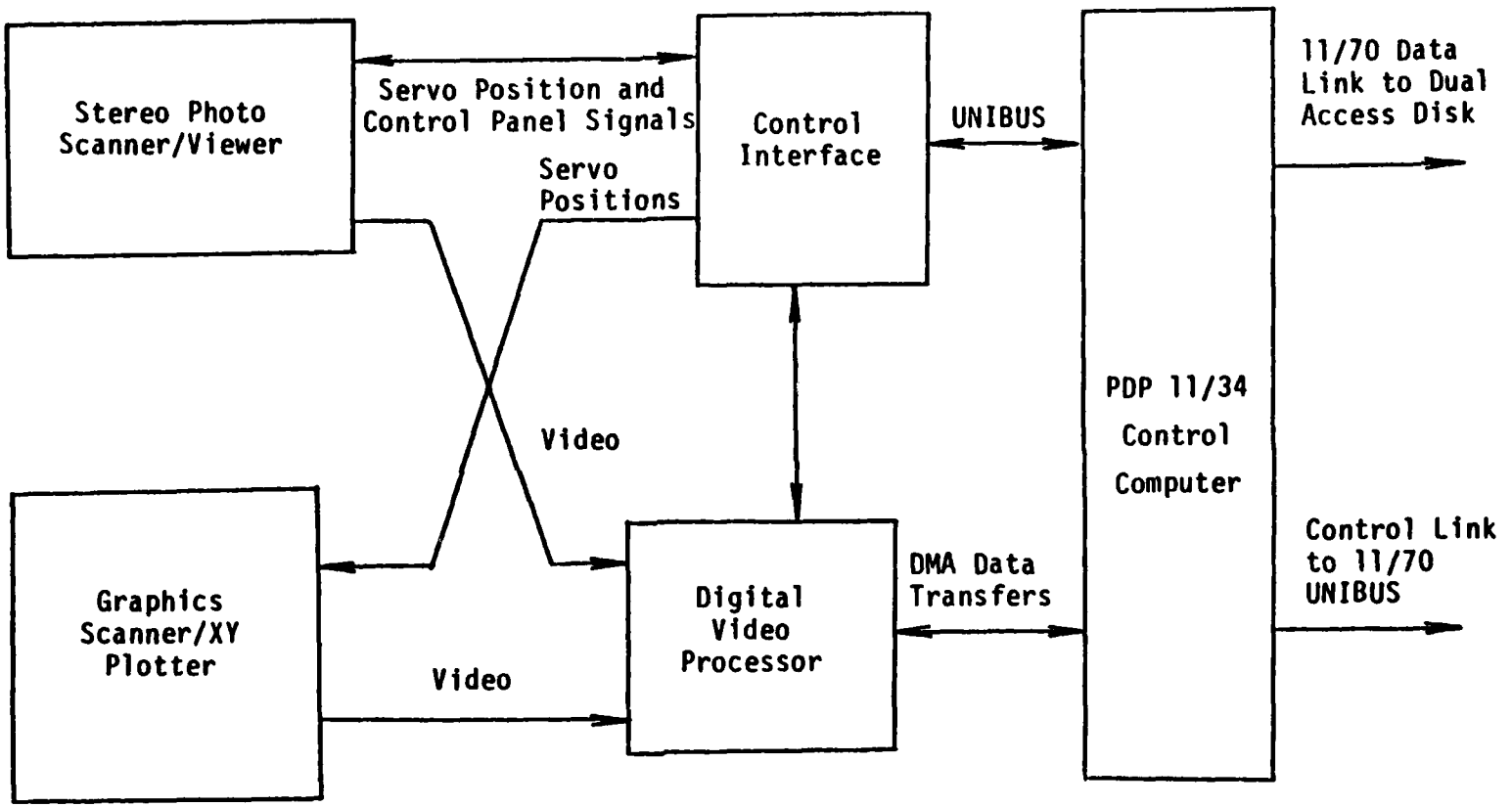


Figure 8-1 - AFES Scanner System Diagram

the raster is 1024 by 1024 pixels. The complete control of photo stage motion and optical rotation allows for correction of skew image distortion while scanning. This feature is particularly useful in scanning panoramic photographs.

The graphics scanner is essentially an XY plotter with a linear array camera mounted on the plotting head. This scanner has fixed optics and scans the map or feature manuscript at a constant magnification. The graphics scanner provides for either front or back illumination of the image. This allows both transparent and opaque manuscripts to be scanned. The scanning technique for the graphics scanner is essentially similar to that of the photograph scanner. The camera is moved in one direction while image lines are sampled along an orthogonal direction. Because of the fixed optics, however, the graphics scanner always produces a rectangular raster format and a fixed resolution of 80 micros.

The video processing and camera control hardware operates in conjunction with the three scanners to produce digitized image data. This hardware provides synchronization and control signals to the detector array camera electronics for image line sampling and serial transfer of image data from the scanner to the video processing hardware. Image information comes from the scanner in the form of sampled analog video. Since only one set of video processing hardware is required for the three scanners, video selection circuitry is provided for connecting any one of the scanners to the video processing hardware. Video processing includes analog-to-digital (A/D) conversion, compensation for sensor dark level, compensation for sensor gain, geometric corrections, and transformation of image intensity data.

The scanner system has its own Digital Equipment Corporation PDP 11/34 control computer which runs under the RSX-11M operating system. This computer controls overall operation of the scanner system, interacts with the operator during setup, and communicates with the AFES main PDP- 11/70 computer. For scanning operation, the PDP-11/70 programs generate commands to initiate

photograph or graphics digitization. Information is passed to the scanner system computer indicating which scanner to select, photograph or graphics coordinates for the center of the scan, magnification to be used, rotation of the array, and scan motion parameters. The computer then generates the necessary servo commands and control commands to scan and digitize an image, read scan line data into the PDP-11/34 memory, and transmit this data to the mass storage disk. Once the process has been initiated, it proceeds independently to scan and store a digital image. During scanning the PDP-11/70 is free to perform other AFES System operations, assuming those operations do not interfere with the storage of digitized image data on the dual access disk unit by the image digitizer system computer.

The scanner system computer handles the real-time transformations from photo coordinates to stage coordinates for scanning and positioning the photographs. These transformations are established at setup time by performing an interior orientation for each of the two photographs. To provide for manual motion of the photographs during interior orientation, the system has a control panel with an incremental input control. This is connected to the computer through the control interface. To permit entry and display of data, a CRT terminal is provided with the computer system. This facilitates entry of orientation data and provides for display of data for monitoring image digitizer system operation.

8.3 PHOTOGRAPH SCANNER/VIEWER UNIT

The stereo photograph scanners are contained in a single unit which rests on an equipment cabinet the size of an office desk (see Figure 8-2). The unit contains two 9 inch by 18 inch servo-controlled photograph XY stages, two line array cameras, zoom optics for both cameras, and stereo viewing optics. The photograph stages are of stage-on-stage design with round ways and ball bushings. Optical encoders quantize X and Y motion to a precision of 2 micros. An accuracy of 5 micros rms is achieved by applying corrections to the servo outputs within the computer.

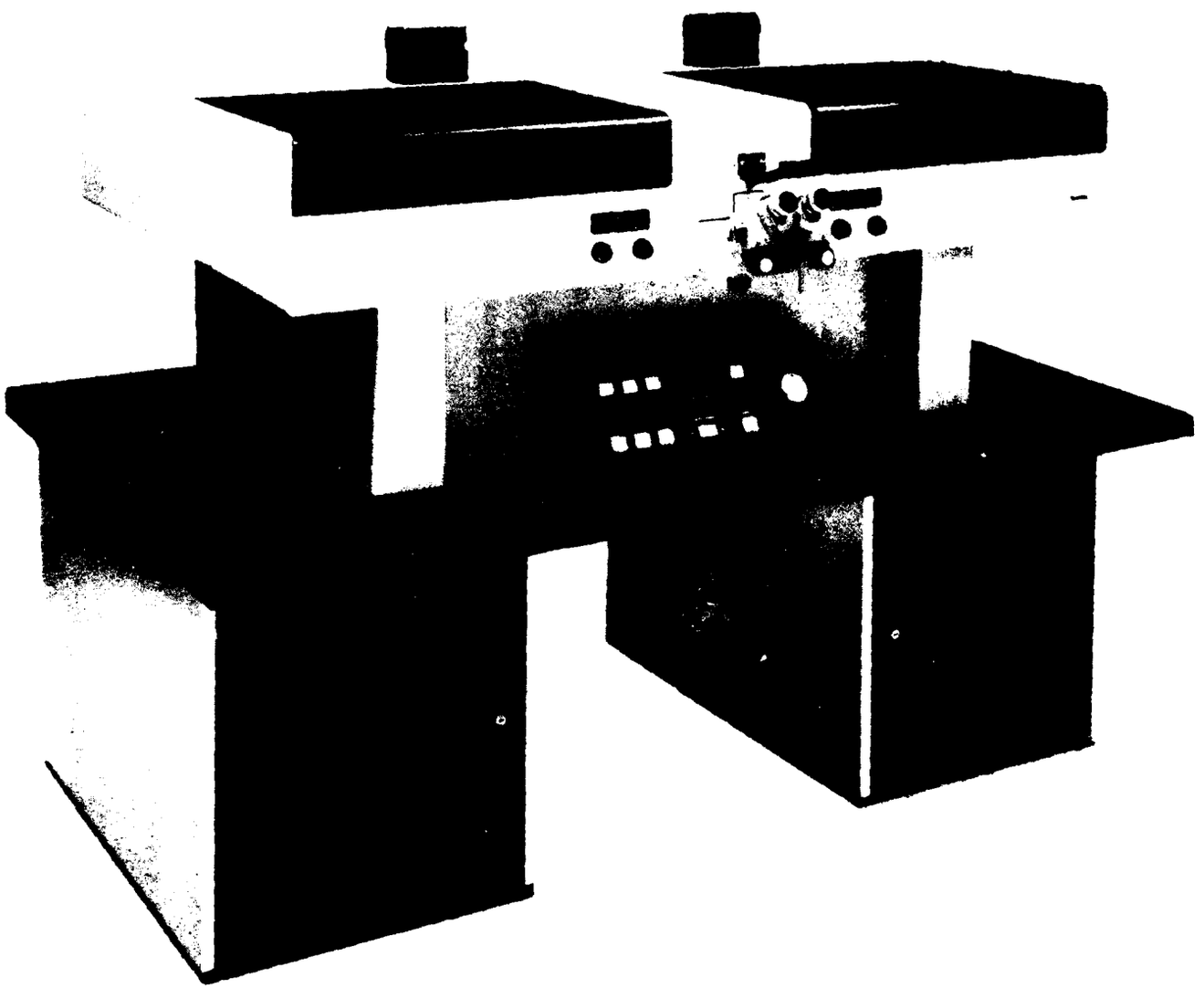


Figure 2-1. 7413 Scanner/Viewer Unit

Servo-controlled zoom optics assemblies provide computer control of scanning magnification over a 6:1 range. The zoom optics are based on a commercial zoom microscope assembly adapted for servo control. The magnification range of the scanner optics is about 0.43X to 2.6X for pixel sizes in the range of 5 micros to 30 micros. The line array camera can be rotated through $\pm$ 20 degrees to permit scanning along the stereo baseline. This rotation is also servo-controlled from the computer.

In addition to the scanning optics, a stereo viewing optical system is provided for performing photograph interior orientation and locating conjugate stereo imagery. The viewing optics consist of the optics from a Bausch and Lomb stereo zoom transfer scope modified for use in the AFES Scanner system. The viewing optics have image rotation and zoom magnification capabilities. Magnification range is from 6X to 40X and maximum field of view is 30 millimeters. Optical reference marks for manually measuring photograph positions are luminous marks multiplexed into the viewing optics at a point immediately below the photograph stages. Photographs are mounted emulsion down on the stage glass and are viewed and scanned from below. Illumination of the photographs is from the top. The illumination lamps are operated from well regulated DC power supplies, one of which can be controlled by the computer. As the scanner magnification is varied, the illumination level is automatically adjusted by the computer to keep the line array camera operating at its nominal output level.

To provide the requisite operator control functions for setup and interior orientation, a control panel is mounted on the photograph scanner/viewer unit in front of the operator. This panel contains a rate input control for slewing the various scanner servos, and pushbuttons to control servo power and for selecting the servo axis to be controlled. A busy lamp on the panel signals the operator when the computer is engaged in performing a scan and digitize operation and cannot respond to operator inputs.

8.4 GRAPHICS SCANNER/XY PLOTTER

The graphics scanner consists of a linear array camera mounted on the drafting head of an XY plotter. The linear array is aligned with the X axis of the plotter and cannot be rotated. Magnification of the scanner is fixed at about 6X. This allows the 13 mm linear array to cover about 80 mm on the map or feature manuscript. The optical axis of the camera is offset a precisely known amount from the stylus chuck of the plotter. This allows the stylus to be used as a reference pointer to the center of the image to be scanned. Scanning is accomplished by moving the plotter at a constant velocity in the Y direction while sampling and reading out image lines in the X direction. The scanner scans and digitizes a complete 1024 by 1024 pixel image in about 8 seconds.

Both transparent and opaque manuscripts can be scanned with the graphics scanner. Front illumination is provided for scanning opaque manuscripts such as maps. For transparent material, the backlighting of the plotting table is used. A portable control panel allows the operator to manually control the plotter servos at the plotter. This is used for set-up of map base or feature graphics on the XY plotter.

8.5 LINE ARRAY CAMERAS

The basic photosensing device of the three scanners is a 1024 element charge-coupled device (CCD) linear array. This device is an integrated circuit containing charge-integrating photo sites and CCD analog shift registers. The photo sites convert photons to electronic charge and integrate the charge during an integration period. To read out the device, the charge packets are all simultaneously transferred (in parallel) to two CCD analog shift registers. The shift registers are then read out serially and their outputs combined to obtain a single video output signal from the device.

In the AFES Scanner System, the array sensors are incorporated in prepackaged linear array camera systems. The camera contains the necessary circuitry for driving the array and sampling its output to produce a sampled analog video signal at the output of the camera. Since the photometric fidelity of the array is not by itself adequate for the AFES system, external video processing circuitry is provided which corrects dark current and gain variations across the array.

The photo sites of the array are 8 x 13 micros elements on 13 micros centers with 5 micros channel stop bands between them. The array of 1024 elements is about 13 mm long. The geometric stability of the array is excellent since the photo sites are fixed on the device substrate.

Dynamic range of the array device itself is typically 500:1. The camera electronics, however, add some noise that reduces this range to a minimum of 200:1, or $\pm$ 1 part in 400. Dynamic range is defined as the ratio of saturation voltage to random output noise (peak-to-peak) on a per-cell basis. Variations in dark current and gain from cell to cell generate noise which is much greater than the random noise, but the dark current and gain effects can be compensated and, therefore, do not limit dynamic range.

8.6 VIDEO PROCESSOR

Video signals from the scanners must be sampled and converted to 8-bit binary data for storage and manipulation in the AFES system. Furthermore, the photometric fidelity of data directly from the scanner, while perhaps suitable for stereo viewing and map display, is not suitable for automatic feature extraction. The individual elements in the sensor array exhibit different dark current noise and sensitivity (gain) characteristics. These variations, while repeatable for each element, are too large to obtain adequate photometric resolution across the array. For images which are to be processed by automatic feature extraction, the video data must, therefore, be further processed to remove dark current and gain variation effects. It is also

desirable to obtain logarithmic as well as linear image data. The A/D conversion, photometric correction, and log/linear transformation are all performed by the video processing subsystem.

A diagram of the video processing subsystem is shown in Figure 8-3. Video signals from the three linear array cameras enter the video select circuitry where one of the three signals is selected as the video source. The selected video is then amplified by a video amplifier to raise the signal level and reject any common mode noise picked up as a result of transmission from the camera to the video processor. The video is then sampled periodically and converted to a 10-bit binary data stream. The sampling is performed synchronously with serial readout of the array, so successive digital data words correspond to outputs of successive sensor elements along the array. If the image is to be used only for viewing, the image data may be stored directly in the output buffer. If the image is to be processed by automatic feature extraction, corrections are made to the individual image data words.

As shown in Figure 8-3, the output of the A/D converter is applied to summing circuitry where an 8-bit value is subtracted to correct for dark current. Each cell of the 1K by 8 random access memory (RAM) contains a dark signal correction value for one element of the sensor array. After dark signal correction, the image data is corrected for variations in gain of the individual elements. This is done by multiplying each data sample by a gain factor. Each cell of a 1K by 9 RAM contains a gain correction factor for one array element.

The output of the multiplier is a stream of 10 bit data words. This data can be either stored directly as 8-bit words in the output buffer or applied to a log transform random access memory (RAM). If the RAM is bypassed, the output data is a linear function of photograph transmission. If the output data is obtained from the transform RAM, a logarithmic response is obtained. Since the RAM can implement almost any mapping, other rather arbitrary functions can be implemented by loading the transform RAM with data for the

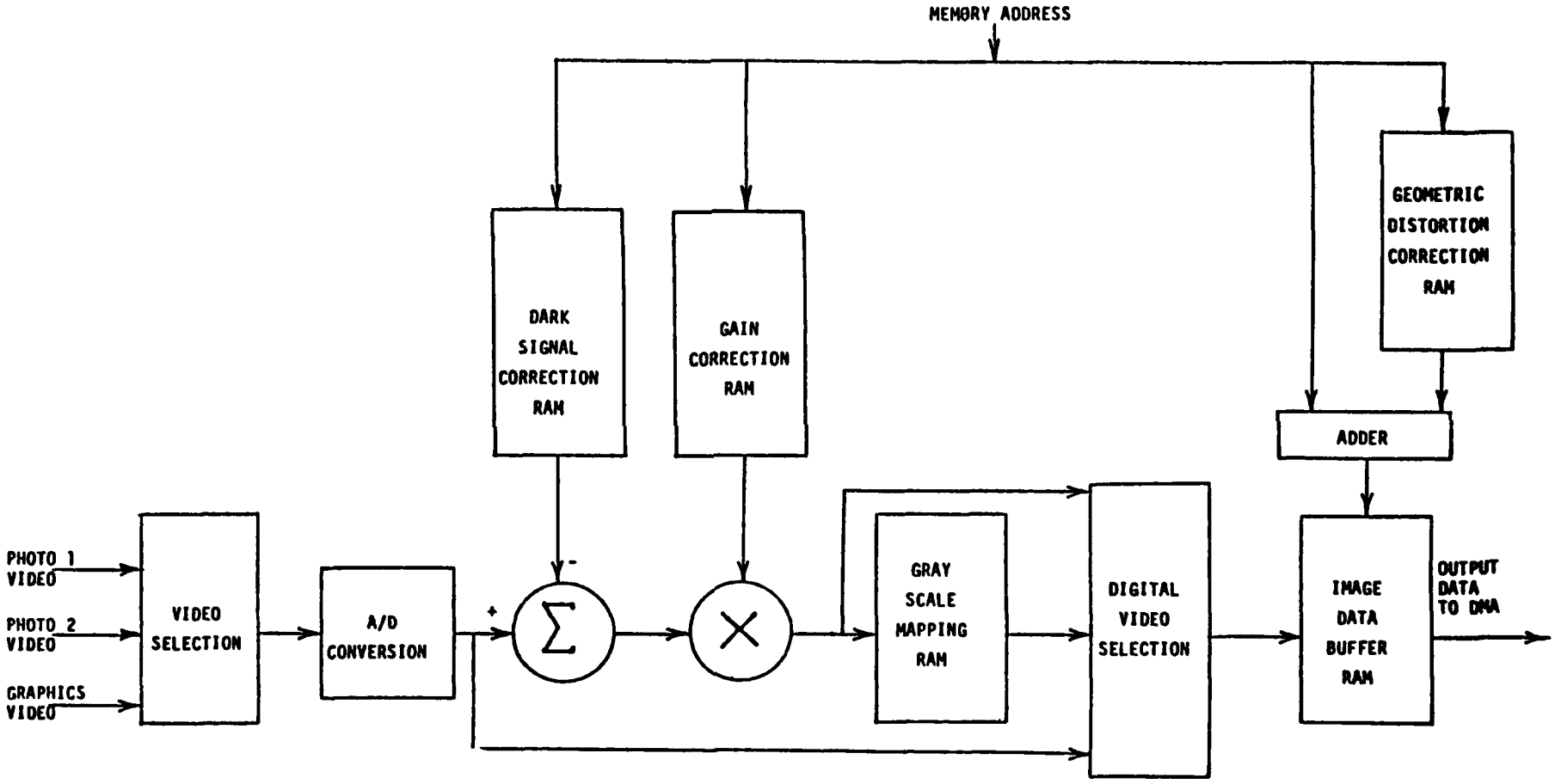


Figure 8-3 - Video Processor

desired transformation.

The output buffer RAM collects the image data during readout and processing of one line of image data. Successive processed image data samples are stored in successive locations in the buffer. When the line is completed, the scanner control computer initiates a direct- memory access (DMA) transfer of the buffer contents into the computer memory. To provide better storage efficiency and higher speed of transfer, the data is packed by transferring two successive 8-bit samples as one 16-bit word over the PDP-11/34 UNIBUS.

Because the optical systems of the scanners introduce geometric distortions, corrections must be made to the image data to obtain good geometric linearity. This is accomplished by applying small corrections to the output buffer memory address references during readout. The corrections are stored in a geometric correction RAM and are added to the memory address of the output buffer RAM. The corrections are only applied when the image data is read out; all other data transfers to and from the buffer RAM use the normal sequential addressing. All RAMs in the video processor can be loaded and read from the computer through the DMA transfer device. The video processor memories can, therefore, be modified during system operation. The video processor can also be tested for proper operation by transferring test data to and from the various RAMs in the video processor. Modification of RAM contents during system operation is used in automatic calibration procedures.

8.7 COMPUTER CONTROL SYSTEM

The control computer and its peripherals comprise a major part of the system. The various control sequences, computations, and data transfers necessary to implement the various functions of the system are performed by the control computer under the direction of its stored programs. The computer is a PDP 11/34 with 32K words of memory and a floating point processor. Peripherals include a CRT terminal, floppy disk unit, DMA transfer interface, RP06 dual access mass storage disk unit, and a special control interface. The

CRT terminal is used primarily for operator interaction with the system. It allows the operator to enter and display data, select modes of operation, and initiate various functions of the system. The floppy disk unit is used to read programs from a permanent storage medium during system startup. The DMA transfer interface is used to transfer blocks of data between the computer memory and the various RAMs of the video processor. The RP06 dual access disk unit is used to store image data for transfer to the PDP-11/70 AFES computer system. During image scanning and digitizing, image data is written into the RP06 on a line-by-line basis. When a complete image is digitized and stored on the disk, the PDP-11/70 computer reads out the data by way of the other disk access port.

The control interface unit contains parallel input/output (I/O) and servo control hardware to allow the computer to control the three scanners and the video processor. Servo control exercised by the computer includes the two axes of each photo stage, the X and Y axes of the graphics scanner, the magnification optics, and the rotation optics. Limit switches on the various servo systems are sensed through the parallel input hardware of the interface. Parallel I/O hardware is used to control and sense status of the video processor. In addition, an operator control panel is serviced through parallel I/O hardware of the interface unit.

The various functional capabilities of the stereo image digitizer system are implemented primarily by the computer programs. These programs respond to inputs from either the operator or the 11/70 computer and perform the various programmed functions of the system. They generate motion commands for the servo travel, and take appropriate remedial action. During the performance of the various system functions, the programs control the sequences of operations and the flow of data through the system.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


3-8
DT